UNIVERSITY of BRADFORD

Faculty of Engineering & Informatics

IMCS International Membrane Computing Society

Proceedings of the 18th International Conference on Membrane Computing (CMC18)

Edited by Marian Gheorghe, Savas Konur and Raluca Lefticaru

24-28

July 2017

Bradford, UK

www.computing.brad.ac.uk/cmc18/

UOB CONFERENCE 2017 WWW.COMPUTING.BRAD.AC.UK/CMC18/

Proceedings of the Eighteenth International Conference on Membrane Computing (CMC18)

24-28 July, 2018

Bradford, UK

Marian Gheorghe
Savas Konur
Raluca Lefticaru
Editors

Proceedings of the 18th International Conference on Membrane Computing (CMC18)

Edited by: Marian Gheorghe, Savas Konur and Raluca Lefticaru

Published in July 2017

Preface

The present volume contains the invited contributions and a selection of papers presented at the 18th International Conference on Membrane Computing (CMC18), held in Bradford, UK from July 24 to July 28, 2017. Further additional information on this conference can be found at the following website: http://computing.brad.ac.uk/cmc18/

The CMC series started with three workshops organized in Curtea de Argeş, Romania, in 2000, 2001 and 2002. The workshops were then held in Tarragona, Spain (2003), Milan, Italy (2004), Vienna, Austria (2005), Leiden, The Netherlands (2006), Thessaloniki, Greece (2007), and Edinburgh, UK (2008).

The 10th edition was organized again in Curtea de Argeş, in August 2009, where it was decided to continue the series as the Conference on Membrane Computing (CMC). The following editions were held in Jena, Germany (2010), Fontainebleau, France (2011), Budapest, Hungary (2012), Chişinău, Moldova (2013), Praha, Czech Republic (2014), Valencia, Spain (2015) and Milan, Italy (2016).

CMC18 has been organized, under the auspices of the International Membrane Computing Society, and the European Molecular Computing Consortium by the Modelling, Testing and Verification Research Group[1], School of Electrical Engineering and Computer Science, the University of Bradford, UK.

CMC18 consisted of two different parts: standard sessions, Tuesday to Thursday, and interactions between participants, Friday. Monday was the arrival day for most of the participants. The standard sessions included invited lectures given by Erzsébet Csuhaj-Varjú (Budapest, Hungary), Harold Fellermann (Newcastle, UK), Michael Fessing (Bradford, UK) and Maciej Koutny (Newcastle, UK).

The editors express their gratitude to the Program Committee, the invited speakers, the authors of the papers, the reviewers, and all the participants for their contributions to the success of CMC18.

The support of the School of Electrical Engineering and Computer Science of the University of Bradford and the Prize for the Best Student Paper awards granted by Springer-Verlag are gratefully acknowledged.

July 2017

Marian Gheorghe
Savas Konur
Raluca Lefticaru

[1] http://www.bradford.ac.uk/ei/electrical-engineering-and-computer-science/research/modelling-testing-and-verification/

Table of Contents

Invited Talks

Simple and Small: On Two Concepts in P Systems Theory (Abstract)

Erzsébet Csuhaj-Varjú

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

Membrane systems or P systems, introduced by Gheorghe Păun in 1998, are distributed computing devices inspired by architecture and functioning of living cells and tissues.

Roughly speaking, a customary P system consists of a membrane structure (a virtual graph, usually a hierarchical arrangement of membranes) and finite multisets of objects which can be found in the compartments. The objects represent biochemical ingredients, the membrane structure describes the inner structure of the cells or the tissues. The compartments are associated with a finite number of rules according to which the objects and the membrane structure can change (evolve) and the objects can be communicated among the (neighbouring) compartments and the environment of the P system. The first models were given by a tree-like structure and symbol-objects, later several other variants of P systems have been introduced and studied.

A lot of research has been devoted to the computational power of different variants of P systems, with special emphasis put on models with restricted size. It has been shown that several types of membrane systems with a very small number of compartments, objects, rules, or other important components are very powerful, even computationally complete computing devices. In this talk, we identify some of the reasons of this fact and discuss the relations between the different size parameters of the same type of P systems or that of different P system variants. We also analyse the concept ”small” in P systems theory and derive some conclusions.

Another term we often find in the literature of P systems is ”simple”. Several approaches to this concept can be and have been considered, starting from syntactic simplicity to simplicity in functioning. In our talk, we analyse some important P system models from these points of view and attempt to provide some general conditions that a P system should satisfy to be called simple. We also discuss the limits of using such simple P systems as computing devices and as modelling tools.

Petri Net Based Synthesis of Tissue Systems (Abstract)

Maciej Koutny

School of Computing Science
Newcastle University
Newcastle upon Tyne, NE1 7RU, United Kingdom

Tissue systems, generalising membrane systems, are a computational model inspired by the functioning of living cells. In particular, they reflect the way in which chemical reactions take place in cells and molecules move from one compartment to another [14, 4]. Reactions are represented by evolution rules that specify which and how many molecules can be produced from given molecules of a certain kind and quantity. Membrane systems model the computational and communication processes within a single cell divided by membranes into compartments; rules belong to compartments and the molecules that are produced either remain in the compartment or can be delivered to a neighbouring (i.e., enclosed or surrounding) compartment. Hence a membrane system has an associated tree-like structure describing the connections that can be used for the transport of molecules. This is generalised in tissue structures to arbitrary graphs allowing communication along all edges. The nodes of the graph associated with a tissue system represent, e.g., cells in a tissue, and the edges are the channels along which molecules are passed. Both membrane and tissue systems are essentially multiset rewriting systems with their dynamic aspects including potential behaviour (computations), deriving from their evolution rules. Consequently, they are similar to Petri nets. In particular, there is a canonical way of translating membrane systems into Petri nets with transitions corresponding to evolution rules [10]. This translation is faithful in the sense that it relates computation steps at the lowest level and induces in a natural way extensions and interpretations of Petri net structure and behaviour. The membrane structure is translated into *localities* associated with transitions. The locality of a transition represents the compartment to which the corresponding evolution rule belongs. The localities of transitions make it possible to define a *locally maximal* step semantics in addition to the more common sequential semantics and (maximal) step semantics. Locally maximal steps model localised synchronised pulses with maximal concurrency restricted to compartments.

Petri nets are a well-established general model for distributed computation [5, 6, 15] with an extensive range of tools and methods for construction, analysis, and verification of concurrent systems. The strong semantical link between the two models invites to extend existing Petri net techniques, bringing them to the domain of membrane systems. An example is the process semantics of Petri nets that can help to understand the dynamics and causality in the biological evolutions represented by membrane systems [8, 10]. More details on the relationship between Petri nets and membrane systems can be found in, e.g., [7, 11].

This talk will focus on the synthesis problem understood as the problem of the algorithmic construction of a system from a specification of its observed or desired behaviour. Automated synthesis from behavioural specifications is an attractive and powerful way of constructing correct concurrent systems [1–3, 13]. The paper [9] considered the synthesis of membrane systems from (step) transition systems, and the paper [12] discussed the same problem for membrane systems. Both papers demonstrated how a solution to the synthesis problem of Petri nets, based on the notion of regions of a transition system, leads to a method for the automated synthesis of membrane systems. The talk will show how the synthesis problem for tissue systems (with locally maximal concurrency) can be solved when the tissue structure of the system to be constructed is given together with the step transition system. Following this, a method for extending the basic solution to cope with situations when the structure of the target tissue system has to be constructed will be presented.

Acknowledgement This talk is based on research conducted in collaboration with Jetty Kleijn, Marta Pietkiewicz-Koutny, and Grzegorz Rozenberg.

References

1. Badouel, E., Darondeau, P.: Theory of regions. In Part I of [15], 529–586
2. Darondeau, P., Koutny, M., Pietkiewicz-Koutny, M., Yakovlev, A.: Synthesis of nets with step firing policies. Fundamenta Informaticae 94 (2009) 275–303
3. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-structures. Acta Informatica 27 (1989) 315–368
4. Păun, G., Rozenberg, G., Salomaa A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
5. Koch, I., Reisig, W., Schreiber F. (eds.): Modeling in Systems Biology — The Petri Net Approach. Springer Verlag (2010)
6. Jensen, K., van der Aalst, W.M.P., Balbo, G., Koutny, M., Wolf, K. (eds.): Transactions on Petri Nets and Other Models of Concurrency VII (2013)
7. Kleijn, J., Koutny, M.: Petri nets and membrane computing. In [4], 389–412
8. Kleijn, J., Koutny, M.: Processes of membrane systems with promoters and inhibitors. Theoretical Computer Science 404 (2008) 112–126
9. Kleijn, J., Koutny, M., Pietkiewicz-Koutny, M., Rozenberg, G.: Membrane systems and Petri net synthesis. MeCBIC. EPTCS 100 (2012) 1–13
10. Kleijn, J., Koutny, M., Rozenberg, G.: Process semantics for membrane systems. Journal of Automata, Languages and Combinatorics 11 (2006) 321–340
11. Kleijn, J., Koutny, M., Rozenberg, G.: Petri nets for biologically motivated computing. Scientific Annals of Computer Science 21 (2011) 199–225
12. Kleijn, J., Koutny, M., Pietkiewicz-Koutny, M.: Tissue systems and Petri net synthesis. Transactions on Petri Nets and Other Models of Concurrency IX (2014) 124–146
13. Kleijn, J., Koutny, M., Pietkiewicz-Koutny, M., Rozenberg, G.: Applying regions. Theoretical Computer Science 658 (2017) 205–215
14. Păun, G.: Membrane Computing, An Introduction. Springer Verlag (2002)
15. Reisig, W., Rozenberg, G. (eds.): Lectures on Petri Nets I & II. Lecture Notes in Computer Science 1491 & 1492. Springer (1998)

Toward Programmable Biology: Techniques for Computing in and with Cells (Abstract)

Harold Fellerman

School of Computing Science, Newcastle University
Claremont Tower, Newcastle University, NE1 7RU, United Kingdom
harold.fellermann@newcastle.ac.uk

TBC

Epigenetic Regulation of Gene Expression in Health and Disease (Abstract)

Michael Fessing

School of Chemistry and Biosciences, University of Bradford
Richmond Road, Bradford, BD7 1DP, United Kingdom
mfessing@bradford.ac.uk

TBC

Regular Papers

Simulating Evolutional Symport/Antiport by Evolution-Communication and vice versa in Tissue P Systems with Parallel Communication

Henry Adorna[1,2], Artiom Alhazov[1,3], Linqiang Pan[1,4], Bosheng Song[1]

[1] Key Laboratory of Image Information Processing
and Intelligent Control of Education Ministry of China,
School of Automation
Huazhong University of Science and Technology,
Wuhan 430074, China
Email:lqpan@mail.hust.edu.cn
[2] Department of Computer Science (Algorithm and Complexity)
University of the Philippines Diliman
1101 Quezon City, Philippines
E-mail: hnadorna@dcs.upd.edu.ph
[3] Institute of Mathematics and Computer Science
Academy of Science of Moldova
Academiei 5, Chişinău, MD-2028, Moldova
Email:artiom@math.md
[4] School of Electric and Information Engineering,
Zhengzhou University of Light Industry,
Zhengzhou 450002, China

Abstract. We aim to compare functionality of symport/antiport with embedded rewriting to that of symport/antiport accompanied by rewriting, by two-way simulation, in case of tissue P systems with parallel communication. A simulation in both directions with constant slowdown is constructed.

Keywords: Membrane computing, Evolution-Communication, Evolutional Symport/Antiport, Simulation

1 Introduction

Membrane systems with symbol-objects are a theoretical framework of parallel distributed multiset processing. Its two essential features are rewriting (also sometimes called evolution) and communication. One extreme case is using rewriting alone, then in the non-cooperative case the computational power is rather weak, while cooperation of two symbols already leads to the computational completeness. To use distributivity, some mechanism of passing information between regions is necessary, and moving objects is most natural choice.

The second extreme case is using communication alone: moving objects without creating, destroying or modifying them. In one of the most studied models the rules are called symport/antiport rules, respectively if objects are moved across a membrane or channel in one/both directions. Clearly, without creating objects, for being able to use more symbols in the computation and/or result than there are initially in the system, some unbounded source of them is needed, e.g. the environment. Note that communication rules across the skin membrane can do the same work as rewriting rules do. It follows that already with a single membrane, communication rules involving up to three objects is already enough for computational completeness, while further restricting rules makes the second membrane necessary, and the proof become much more complicated.

The historically first model of membrane computing is the transitional one: communication is embedded into rewriting by target indications for objects in the right hand side of rules. While not increasing the computational power, it brings additional benefits from the structure, for example, generating languages is considered.

A different approach is to allow both rewriting (which may be restricted to be non-cooperative) and communication rules (no longer needing unbounded object supply in the environment). The model is called evolution-communication. It allows computational completeness constructions already with communication rules of up to two objects, and the proofs are simpler than in pure symport/antiport case.

Finally, a model embedding rewriting into communication rule has been recently introduced in [8], called evolutional symport/antiport. It has been shown to either reach the computational completeness or efficiency (i.e., solving intractable problems in polynomial number of steps) with smaller bounds on the size of rules, or improve existing symport/antiport results due to a more refined complexity measure, accounting for both the number of reactants and number of products.

Overall in the literature on membrane computing, a huge number of results are those establishing computational completeness of some model or variant by simulating another model, which is usually sequential. However, simulating P systems (except the classes P systems having considerably less power than the computational completeness) by something (significantly different P systems or completely different models of computation) are much more rare, and one of the reasons for it is immediate: implementing maximal parallelism itself is considerably more tedious than establishing the computational completeness. We can mention two example appearing in the literature: simulating a two-membrane proton pumping system (a special particular case of evolution-communication P system) with one proton by a P system with one bi-stable catalyst [1] and simulating spiking neural P systems with delays by those without delays [2]. However, the first one is a one-way simulation, and the latter is a simulation staying within the same model. Imagine that requiring the slowdown to be bounded by a constant would pose a significant additional difficulty. Yet, this is the kind of question to be addressed in this paper.

There was a discussion about how evolution-communication relates with evolutional symport/antiport; even though both models are known to be computationally complete, the proofs are rather different. Hence, it presents an interest to simulate by one model the **process** of computation in the other model and vice versa. To keep simulation transparent and "nice," we impose a condition that the simulation slowdown must be (limited by a) constant, and the configurations of the simulated system should be obtainable from the configurations of the simulating system in some easy way, e.g., a morphism. Note that we allow the simulation to be incorrect, as long as the corresponding computation of the simulating system is not halting. As it will be clear later, it is easily decidable which steps of simulation are incorrect, e.g., by checking for appearance of the special symbol #.

We should note that due to the nature of the problem, we assume two features of the models: tissue structure and parallel communication; we now explain the reasons for that. First, in evolution-communication model objects may be massively renamed in parallel; to be able to simulate this with a constant slowdown, communication rules must also be applicable in a massively parallel way. Second, without the tissue structure it would be very difficult to synchronize evolution with communication. Indeed, in pure symport systems the information is propagated such that the signal would return to the same region in even number of steps, while it could reach the neighboring region in odd number of steps. Using antiport, circumventing this problem does not seem easy either, since we assume the rules my be simulated in massively parallel way.

This paper is organized as follows: the necessary definitions are given, then some easy cases of the problem formulation are solved. Next, evolution-communication P systems are simulated by evolutional symport/antiport. The converse direction in general is more involved. After some preliminary arguments, the solution is incrementally constructed. First, the case of simulating one rule without idle objects is solved, also handling incorrect assignment of objects to rules. Second, idle objects are handled, also verifying the maximality of parallelism. Third, halting is approached within the same model. Finally, a complete solution is given in tables in the end of the paper. Having accomplished the goal and formulated the results, we give concluding remarks.

2 Definitions

This paper will be dealing with simulating evolution, and communication rules within the framework of tissue P systems with parallel communication. In particular, the well known evolution-communication rules are compared to the so-called evolutional symport/antiport rule introduced in [8]. We provide here only essential definitions that would be needed in the development of the result.

The basic P system structure that we will consider in doing simulations in both direction is the so-called tissue P systems, first considered in [7]. We now recall their definition, keeping in mind that we do not use states of cells, that the membrane channels can be easily deduced from the rules, and that the set of symport/antiport rules (evolutional or not) is global.

We denote k-th symbol of string u, $1 \leq k \leq |u|$, by $u[k]$.

Definition 1 Tissue P Systems *[7] A* tissue P system *or* tP system, *of degree $m \geq 1$, is a construct*

$$\Pi = (O, w_1, w_1, \ldots, w_m, R, i_{out}),$$

where:

1. *O is a finite non-empty alphabet (of objects);*
2. *$i_{out} \in \{1, 2, \ldots, m\}$ indicates the output cell;*
3. *w_i specifies the initial multiset of objects in cell i, $1 \leq i \leq m$;*
4. *R is a finite set of communication rules.*
5. *Additionally, in the evolution-communication model, sets $R_1, \ldots, R_m$ of rewriting rules should also be specified, the subscript of the set indicating the cell.*

The definition/representation of symport and antiport rules is made in a different manner from the way it was originally introduced in the literature [3], because the underlying structure is a graph. In particular, by $(\, i,\, u,\, j\,)$, we mean from cell i a multiset of objects represented by u is sent to cell j; and $(\, i,\, u\, /\, v,\, j\,)$, means that cell i brings u to cell j, while cell j brings v to cell i at the same time.

Definition 2 Evolution-Communication Rules

1. **Evolution rule:** *$r\colon a \to u$, where $a \in O$, and $u \in O^*$. r is non-cooperative.*
2. **Symport rule:** *$(\, i,\, u,\, j\,)$, where u represents a multiset of symbols from O. The* **length** *of a symport rule is equal to $|u|$.*
3. **Antiport rule:** *$(\, i,\, u\, /\, v,\, j\,)$, where u and v represent multisets of symbols from O. The* **length** *of an antiport rule is equal to $|u| + |v|$.*

The following definition is a system with rules introduced in [8] via the following P system variant, excluding cell division:

Definition 3 [8] *A* **tissue** *P system (of degree $q \geq 1$) with* **evolutional symport/antiport** *rules is a tuple*

$$\Pi = (\Gamma, E, M_1, M_2, \ldots, M_q, R, i_{out}),$$

where

1. *Γ is an alphabet of objects.*
2. *$E \subseteq \Gamma$ is a set of objects initially located in the environment in unboundedly many copies.*
3. *M_i, $1 \leq i \leq q$ is a finite multiset over Γ.*
4. *R is a finite set of evolutional communication rules.*
5. *$i_{out} \in \{1, 2, \ldots, q\}$.*

In our purposes, we explicitly state the evolutional symport/antiport rules as follows:

Definition 4 Evolutional Symport/Antiport Rules [8]

1. **Evolutional symport rules:** $[u]_i[\,]_j \rightarrow [\,]_i[u']_j$, *for* $1 < i \leq q, 0 < j \leq q, i \neq j; u \in \Gamma^+, u' \in \Gamma^*$ *or* $i = 0, 1 < j \leq q; u \in \Gamma^+, u' \in \Gamma^*$, *and if* $i = 0$, *then* u *contains at least one object* $a \in \Gamma \setminus E$*;*
2. **Evolutional antiport rules:** $[u]_i[v]_j \rightarrow [v']_i[u']_j$, *where* $0 \leq i \neq j \leq q$, $u, v \in \Gamma^+, u', v' \in \Gamma^*$.

We recall that either model operates in the usual maximally parallel mode: at each step, a non-extendable multiset of rules is chosen and applied. We call an object *idle* if no rule has been assigned to it, and it is carried over to the next configuration unchanged. Clearly, no rule may be applicable to all idle objects.

The typical assumption for tissue P systems is that at most one communication rule may be applied for each channel (i.e., for any unordered pair (i, j) of cells). Throughout this paper we consider tissue P systems *with parallel communication*, meaning that each channel works in the maximally parallel way, similarly to the typical functioning of symport/antiport in cell-like P systems.

Definition 5 Simulation

We say that a rule A *is* **simulated by** *a set of rules* B, *if there exists an injective morphism* h *from configurations of a simulated system into configurations of a simulating system such that for any configuration* x, *if* $A(x)$ *denotes the output of applying rule* A *on* x, *there exist applications of rules in* B, *possible taking multiple steps, such that their output* $B(h(x))$ *on input* $h(x)$ *equals* $h(A(x))$. *Moreover, any halting computation of the simulating system starting with* $h(x)$ *should correspond, in the manner described above, to a halting computation of the simulated system starting with* x.

3 Unrestricted Cases are easy

It is not hard to see that if rewriting in the evolution-communication model is not required to be non-cooperative, then evolutional symport/antiport can be simulated by rewriting alone. This is a particular case of evolution-communication. The rule of the form $[\,u\,]_i[\,v\,]_j \rightarrow [\,v'\,]_i[\,u'\,]_j$ could be converted into rule of the form $h_i(u)h_j(v) \rightarrow h_i(v')h_j(u')$, where $h_k(a) = a_k$ for $a \in O$, $0 \leq k \leq m$; this procedure appears many times in the literature, and usually referred to as *"flattening"*.

Moreover, if we allow the underlying structure of tissue P systems to allow self-loops (note that throughout this paper, we assume that all cells have different labels), then the converse simulation is trivial. Indeed, standard symport/antiport is a particular case of evolutional symport/antiport, while evolution rule $a \rightarrow u \in R_i$ would correspond to an evolutional symport rule on a loop: $[\,a\,]_i\,[\,]_i \rightarrow [\,]_i\,[\,u\,]_i$.

In the rest of the paper, we follow standard assumptions: rewriting is restricted to the non-cooperative case, and self-loops are not allowed.

4 Evolution-Communication via Evolutional Symport/Antiport

Theorem 1 *Let Π be a tissue P system with non-cooperative evolution and parallel communication rules. Let $r\colon a \to u$ be a rewriting/evolution rule in Π. Then there exist three* (3) *evolutional symport/antiport rules that simulate r.*

Moreover, the simulation needs rule of size at most $1 + |u|$, for a rewriting rule of size $1 + |u|$.

Proof. We use the following three (3) evolutional symport/antiport rules and three cells/region, namely, i, i', and i'' in the simulation: The application of the rule is sequential.

$$[\,a\,]_i[\,]_{i'} \to [\,]_i[\,r\,]_{i'},\ [\,r\,]_{i'}[\,]_{i''} \to [\,]_{i'}[\,r\,]_{i''},\ [\,r,]_{i''}[\,]_i \to [\,]_{i''}[\,u\,]_i.$$

Note that the size of $a \to u$ is $1 + |u|$. Clearly, the maximal size of our evolutional symport is $|u| + |a| = |u| + 1$.

Theorem 2 *Let Π be a tissue P system with non-cooperative evolution and parallel communication rules. Let $r\colon (\,i,\, u/v,\, j\,)$ be an antiport rule in Π. Then there exist five* (5) *evolutional symport/antiport rules that simulate r.*

Moreover, the simulation needs rule of size at most $|u|+|v|+2$ for an antiport rule of size $|u| + |v|$.

Proof. The following five (5) evolutional symport/antiport rules could simulate $r\colon (\,i,\, u/v,\, j\,)$:

1. $[\,u\,]_i[\,v\,]_j \to [\,r'\,]_i[\,r''\,]_j$
2. $[\,r''\,]_j[\ ]_{j'} \to [\ ]_j[\,r''\,]_{j'}$, $\quad [\,r''\,]_{j'}[\ ]_i \to [\ ]_{j'}[\,v\,]_i$.
3. $[\,r'\,]_i[\ ]_{i'} \to [\ ]_i[\,r'\,]_{i'}$, $\quad [\,r'\,]_{i'}[\ ]_j \to [\ ]_{i'}[\,u\,]_j$.

The simulation could be done in three (3) steps using five (5) appropriate rules of the simulating system. Also, the simulation needs rule of size at most $|u| + |v| + 2$ for an antiport rule of size $|u| + |v|$.

Corollary 1 *Let Π be a tissue P system with non-cooperative evolution and parallel communication rules. Let $r\colon (\,i,\, u,\, j\,)$ be a symport rule in Π. Then there exist three* (3) *evolutional symport/antiport rules that simulate r.*

Moreover, the simulation needs rule of size at most $|u| + 2$ for an symport rule of size $|u|$.

Proof. It is easy to see that any symport rule can be simulated as a degenerate case of antiport.

5 Evolutional Symport/Antiport via Evolution-Communication

First, we look at the following example, before providing the results of this section:

Example 1 *Let us consider the following evolutional symport/antiport rules of a particular P system Π below.*

$$[\,ab\,]_1[\,]_2 \to [\,]_1[\,x\,]_2,\ [\,ac\,]_1[\,]_2 \to [\,]_1[\,y\,]_2,\ [\,bc\,]_1[\,]_2 \to [\,]_1[\,z\,]_2,$$

where $a, b, c, x, y, z \in O$.

Let $a^2b^2c^2$ be found in cell 1 of Π. Then in a single step, the objects $a^2b^2c^2$ can be transformed into xyz in cell 2.

Let us try simulating these rules in a P system with evolution-communication rules where evolution rules are restricted to be non-cooperative.

Observation 1 *If we do first, communications, then we would have moved $a^2b^2c^2$. Since rewriting is non-cooperative, we end up with an even number of copies of all objects. Thus, we fail.*

Observation 2 *Rewriting is also needed after the communication. In the case, when the right hand side of the rule is shorter than that of the left hand side of the rule, before some symbols are removed, the communication rules must verify their correspondence to the other objects of the rule.*

Hence, at least three steps are necessary for the simulation. Since evolution is non-cooperative, non-determinism seems to be unavoidable.

Observation 3 *We note that neither evolution-communication model nor evolutional symport/ antiport model needs the supply of objects in the environment. On the other hand, since we need rewriting in all working regions, we assume that the environment is not present (or we replace it by a new cell).*

5.1 Simulating a rule with no idle objects

Let us now consider each rule $r\colon [\,u\,]_i[\,v\,]_j \to [\,v'\,]_i[\,u'\,]_j$ of the simulated system.

Assume that the objects at the left hand side of each rule are ordered, that is, given by strings. Then we provide the simulating system with rules $u[k] \to r_k \in R_i$ of region i. These rules rewrite objects represented by $u[k]$, for each position k, $1 \leq k \leq |u|$ of the string u. Similarly, we provide region j of the simulating system the same kind of rule, that is, $v[k] \to r'_k \in R_j$, for each k, $1 \leq k \leq |v|$.

In the next step, the simulating system performs the following antiport (communication) rule $(\,i,\, r_1 \cdots r_{|u|} \,/\, r'_1 \cdots r'_{|v|}\,, j\,)$. This rule allows objects $r_1 \cdots r_{|u|}$ and $r'_1 \cdots r'_{|v|}$ to be sent to regions j and i, respectively, in one step.

Finally, the simulating system will do the final rewriting rules in regions i and j to complete the simulation. In particular, we will have $r'_1 \to v'$, $r'_k \to \lambda \in R_i$, $2 \leq k \leq |v|$, and $r_1 \to u'$, $r_k \to \lambda \in R_j$, $2 \leq k \leq |u|$, respectively.

Note that the above construction suffices alone only if the objects are correctly assigned to the rules and no object remains idle.

We summarize this construction as follows:

Proposition 1 *Let Π be a tissue-like P system with evolutional symport/antiport rules without idle objects appearing in reachable configurations. An evolutional antiport rule can be simulated with evolution-communication (antiport) rules.*

Clearly, evolutional symport can be simulated as a degenerate case of evolutional antiport. We let one of the u or v be empty (string). Thus we have

Corollary 2 *Let Π be some tissue-like P system with evolutional symport/ antiport without any object remaining idle.*

Remark 1 To handle objects that are incorrectly assigned to the rules, we add the following trap rules: $r_k \to \#, \# \to \# \in R_i$, for $1 \leq k \leq |u|$ and $r'_k \to \#, \# \to \# \in R_j$, for $1 \leq k \leq |v|$.

5.2 Simulations with idle objects

Idle objects are those objects in a region or cell that are not supposed to be evolved or communicated yet in a particular moment. These objects must wait until they are allowed to evolve or be communicated by the system, or until the system halts.

Observation 4 *We conclude that in the first simulation step, each object nondeterministically decides between evolving and staying idle. This adds the following rules: $a \to a_0 \in R_i, (\, i, a_0, i' \,)$, $a_0 \to a \in R_{i'}$, $(\, i', a, i \,)$, $a \in O$, $1 \leq i \leq m$, where m is the number of cells in the system being simulated and O is its alphabet.*

But the simulated system is not asynchronous, rather it is maximally parallel.

We proceed with the construction that would also verify that the parallelism of applied rule is maximal.

In order to consider maximality of parallelism during the simulation, we use a technique we call *technique of pairs of objects*. In this technique, one of the objects would be used to test the needed condition (such as absence of something), while the other one is for verifying that the first object passed the test. Thus, the rules we had for the idle objects would now be: $a \to a^{(i)}a_0 \in R_i$, $a_0 \to a_1 \in R_i$, $(\, i, a^{(i)}a_1, i' \,)$, $a_1 \to a \in R_{i'}$, $(\, i', a, i \,)$, $a \in O$, $1 \leq i \leq m$. Note that we could have a rule erasing $a^{(i)}$ in region i', but it is not necessary.

After applying these rules, objects $a^{(i)}$ wait for one step. We use this time to test that no rule should be applicable to the objects are chosen to be idle:

$(i, h^{(i)}(u)/h^{(j)}(v), j)$, where $h^{(k)}(a) = a^{(k)}$, $a \in O$, $1 \leq k \leq m$ define the corresponding morphism.

In the case that there was any applicable rule which was not chosen, we could force to disregard such computation by the following rules: $h^{(i)}(a) \to \# \in R_j$ and $h^{(j)}(a) \to \# \in R_i$.

Note that the simulation of one step for the idle objects takes five steps. To synchronize the simulation of rule applications we add two more steps. Hence, the rules $r_1 \to u'$, $r'_1 \to v'$ are replaced by $r_1 \to (u', 2)$, $r'_1 \to (v', 2)$ where $(\cdot, 2)$ is a morphism naturally defined on O. Also, we add rules $(a, 2) \to (a, 1)$, $(a, 1) \to a$ in each R_k, $1 \leq k \leq m$ for each $a \in O$. This ends process of simulation and we summarize it in the following statement:

Theorem 3 *Let Π be a tissue-like P system with evolutional symport/antiport rules with no objects remaining idle. There exist evolution- communication model that handles such idle objects in maximally parallel manner.*

Corollary 3 *There is an evolution-communication system that handles idle objects of a tissue-like P system with evolutional symport/antiport rules.*

Proof. We replace the rules for the idle objects with the following rules: $a \to a^{(i)}a_0 \in R_i$, $(i, a_0, i'), a_0 \to a^{(i)} \in R_{i'}$, $(1, a^{(i)} / a_1, i')$, $a_1 \to a \in R_i$.

Note that unless the simulated system halts with all the regions being empty, the simulating system never halts.

At this point we would like to mention two “cheating” possibilities to avoid further complexity. The first one is to define for the simulated system, in case of no applicable rules the next configuration to be the same as the current one, and redefine halting as repeating the configuration after a specified number of steps, replacing $\# \to \#$ by $\# \to \#\#$. The second possibility is to globally produce specific additional objects in simulating the application of rules, erased after one step, and use them as promoters to continue the computation. However, we are interested in staying within the same model: classical definition of maximal parallelism and halting and no additional features.

5.3 To halt or not to halt

So far, in the first step of the simulation, each object had two alternatives; to be used in some rule (possibly having choice between multiple rules), or to stay idle; with verification that no rule is further applicable to the idle objects and that the rule assignment is correct. Now, these objects should have a third alternative: **to halt.** Indeed, recall that our goal is a simulation with a slowdown by a factor of constant, and the population of objects in unbounded.

On objects choosing between these three alternatives, we need to verify the following additional conditions. First, either all objects choose halting, or none. Second, no rule should be applicable to the “halting” objects. Third, none of the objects should choose to be “idle, but not halting” if no rule is applicable in the whole system.

Observation 5 *The second condition is similar to that for the idle objects. The first one could be implemented by the pairs technique. The third condition is the most difficult. We verify it with the help of one additional control object in the system.*

We proceed by listing the following rules for the simulating system:

Applying a rule:
We replace $u[k] \to r_k \in R_i$ by $u[k] \to r_k e e_0 \in R_i$, if $i \neq 1$, and if $i = 1$, by $u[k] \to r_k(e,1) \in R_1$.
Producing witnesses of rule applications throughout the system:
Add rules $(e,1) \to ee_0 \in R_1$, $(\,i,\, e,\, 1\,)$, $(\,i,\, e_0,\, 1\,)$, $e_0 \to e_1 \in R_1$, $(\,1,\, ee_1,\, 1'\,)$.
Control object:
(will halt)

$$I_0 \to I_1 \in R_1,\ I_1 \to I_2 I \in R_1,\ I_2 \to I_3 \in R_1,\ (\,1,\, I_3 I,\, 1'\,).$$

(continue the computation)

$$I \to I_4 \in R_2,\ (\,2,\, I_4,\, 1\,),\ (\,1,\, I_3 I_4,\, 1'\,), I_4 \to I_0 \in R_1,\ (\,1,\, I_0,\, 1\,)$$

Notice that object e returns to region 1 from region 2 and moves with an extra object e_1 to region $1'$ by the previous rule.
Checking for absence of "idle but not halting objects."

$$I \to f^{(1)} f_0^{(1)} \cdots f^{(m)} f_0^{(m)} \in R_1,\ (\,1',\, f^{(i)},\, i\,),\ (\,1',\, f_0^{(i)},\, i\,),$$
$$f_0^{(i)} \to f_1^{(i)} \in R_i,\ (\,i,\, f^{(i)} f_1^{(i)},\, 2\,)$$

will be done in the seventh step of the simulation, in each region i by the idle object $f^{(i)}$.
Idle objects:
Replace $a \to a^{(i)} a_0 \in R_i$ by $a \to a^{(i)} a_{-5} \in R_i$, adding rules $a_k \to a_{k+1}$, $-5 \leq k \leq -1$.
Add rules $(\,i,\, a^{(i)} f^{(i)},\, 2'\,)$, $f_1^{(i)} \to \# \in R_i$.
Halting objects:
Add rules $a \to a_h^{(i)} \in R_h$.
Checking inapplicability:

$$(\,i,\, H^{(i)}(u)/H^{(j)}(v),\, j\,), \text{ where } H^{(k)}(a) = a^{(k)},\ a \in O,\ 1 \leq k \leq m,$$

define the corresponding morphisms.
In case there was any additionally applicable rule which was not chosen, rules $H^{(i)}(a) \to \# \in R_j$ and $H^{(j)}(a) \to \# \in R_i$ will force such computations to be disregarded.
Checking absence of $a_h^{(i)}$ in regions i by both rule applications and by "idle but not halting" objects:
Add $g^{(1)} g_0^{(1)} \cdots g^{(m)} g_0^{(m)}$ to the right sides of the rule
$a \to a^{(i)} a_{-5} \in R_i$, $u[k] \to r_k e e_0 \in R_i$, $i \neq 1$ and $u[k] \to r_k(\,e,\, 1\,) \in R_1$.
Add rules

$$(\,i,\, g^{(k)},\, k\,), (\,i,\, g_0^{(k)},\, k\,),\ g_0^{(i)} \to g - 1^{(i)} \in R_i,$$
$$(\,i,\, g^{(i)} g_1^{(i)},\, 2'),\ (\,i,\, g^{(i)} a_h^{(i)},\, 2'\,),\ g_1^{(i)} \to \# \in R_i.$$

Now, if we put together all these rules that we listed for the systems simulating evolutional symport/antiport, see also the simulation synchronization tables, we would have the following results:

Theorem 4 *An evolutional symport/antiport rule on a tissue-like P system with parallel communication could be simulated by evolution-communication symport/antiport rule with constant slowdown.*

Finally, we give our main result:

Theorem 5 (Main Results)
In a tissue P system with parallel communication and non-cooperative evolution rules, we have.

1. *An evolutional symport/antiport rule simulates evolution-communication symport/antiport rule.*
2. *An evolutional symport/antiport rule could be simulated by evolution-communication symport/antiport rule.*

Moreover, the simulation in both directions is within a constant slowdown.

6 Concluding Remarks

We have constructed a simulation of evolutional symport/antiport rule by evolution-communication rules and also, evolution-communication rule being simulated by a system with evolutional symport/antiport rules. We restricted our systems to be tissue P systems with non-cooperative evolution rules and performing parallel communications. The construction is rather challenging and involved, if not very difficult in one direction, but fairly easy in the other direction. Additionally, we presented simulations in both directions with constant slowdown.

We have recalled previous results that provided results relating some model of P systems to another one; transition P systems in evolution–communication P systems with energy [4, 5], and transition P Systems in weighted SN P Systems [6], among others. As we have commented earlier, these are mostly one-way simulation of one model by another. These one-way simulations suggests that there is some homomorphism between these P systems involved. And that under this homomorphism, one could investigate the capability of the simulated system with respect to the properties of the simulating systems under such homomorphism.

In this paper, we somehow suggest that we could have a stronger relation with respect to some homomorphism between these systems. However, we focused on simulating rules of the system itself. It may not be hard to notice that corollary to some simulation results reported in the literature, same analysis as we did in this paper, could be obtained from their construction, say in [1, 2, 4–6], among others.

Our result could spring board some ideas for further investigations:

1. Since we could somehow establish a two-way simulations of rules from different P systems, it might be nice to ask: how could we define the idea of isomorphic P systems?

2. Since we introduce a two-way simulations of rules that allow constant slowdown, could we suggest to have created an idea of a *"reasonable'* reduction scheme for P systems.
3. Since, we have somehow suggested an idea to define "reducibility" in P systems, we might want to realize some complete problems in P systems, also.

7 Acknowledgments

This work is generously supported by National Natural Science Foundation of China (61320106005 and 61033003) and the Innovating Scientists and Technicians Troop Construction Projects of Henan Province (154200510012)

References

1. Alhazov, A.: Number of Protons/Bi-stable Catalysts and Membranes in P Systems. Time-Freeness. In: Freund, R., Păun Gh., Rozenberg G., Salomaa A. (eds) Membrane Computing. WMC 2005. Lecture Notes in Computer Science, vol 3850. Springer, Berlin, Heidelberg (2006), 79-95.
2. Cabarle, F.G.C., Buño, K.C., Adorna, H.N.: Time after Time: Notes on Delays in Spiking Neural P Systems. In: Proceedings of Theory and Practice of Computation: 2nd Workshop on Computation: Theory and Practice. Springer Japan (2013), 82 - 92.
3. Cavaliere, M.: Evolution–Communication P Systems.n: Păun Gh G., Rozenberg G., Salomaa A., Zandron C. (eds) Membrane Computing. WMC 2002. Lecture Notes in Computer Science, vol 2597. Springer, Berlin, Heidelberg (2003), 134-145
4. Juayong, R.A. B, Adorna, H. N.: On simulating cooperative transition P systems in evolution–communication P systems with energy. Natural Computing (2016), 1-11.
5. Juayong, R.A.B., Adorna, H.N.: Relating Computations in Non-cooperative Transition P Systems and Evolution-Communication P Systems with Energy. Fundamenta Informaticae vol. 136(3), (2015), 209-217.
6. Juayong,R.A.B., Hernandez,N.H.S., Cabarle,F.G.C, Adorna,H.N.; A Simulation of Transition P Systems in Weighted Spiking Neural P Systems; In: Nishizaki, S. et al, (eds) Proceedings of Workshop on Computation: Theory and Practice 2013 (WCTP 2013), World Scientific, (2014), 62-78.
7. Martín-Vide, C., Păun, Gh.,Pazos, J., Rodriguez-Patón, A.: Tissue P Systems, Theoretical Computer Science 296 (2003), 295-326
8. Song, B. Zhang, C., Pan, L.: Tissue-like P systems with evolutional symport/antiport rules, *Information Science* 378 (2017), 177-193.

Simulation Synchronization Table

$$F = f^{(1)} f_0^{(1)} \cdots f^{(m)} f_0^{(m)},\ G = g^{(1)} g_0^{(1)} \cdots g^{(m)} g_0^{(m)}$$

Step	Evolve	Idle, not halting	Halting	Control	G	ee_0	F
1	$u[k] \to r_k ee_0 G \in R_i, i \neq 1$ $u[k] \to r_k(e, 1)G \in R_i, i = 1$ $v[k] \to r'_k ee_0 G \in R_j, j \neq 1$ $v[k] \to r'_k(e, 1)G \in R_j, j = 1$	$a \to a^{(i)} a_{-5} G \in R_i$	$a \to a_h^{(i)} \in R_i$	$I_0 \to I_1 \in R_1$			
2	$(i, r_1 \cdots r_{\lvert u\rvert}/r'_1 \cdots r'_{\lvert v\rvert}, j)$ $r_k \to \# \in R_i$ $r'_k \to \# \in R_j$	$a_{-5} \to a_{-4} \in R_i$ $(i, h^{(i)}(u)/h^{(j)}(v), j)$	$(i, H^{(i)}(u)/H^{(j)}(v), j)$	$I_1 \to I_2 I \in R_1$	$(i, g^{(k)}, k)$ $(i, g_0^{(k)}, k)$ may be skipped, if $i = k$	$(i, e, 1)\, i \neq 1$ $(i, e_0, 1), i \neq 1$ $(e, 1) \to ee_0 \in R_1$	
3	$r_1 \to (u', 7) \in R_j$ $r_k \to \lambda \in R_j, 2 \leq k \leq \lvert u\rvert$ $r'_k \to (v', 7) \in R_i$ $r'_k \to \lambda \in R_i, 2 \leq k \leq \lvert v\rvert$ $\# \to \#$	$a_{-4} \to a_{-3} \in R_i$ $a^{(i)} \to \# \in R_j, i \neq j$	$a_h^{(i)} \to \# \in R_j, i \neq j$ $a_h^{(j)} \to \# \in R_i, i \neq j$	$I_2 \to I_3 \in R_1$ $(1, eI, 2)$	$g_0^{(i)} \to g_1^{(i)} \in R_i$ $(i, g^{(i)} a_h^{(i)}, 2')$ may be done one step before	$e_0 \to e_1 \in R_1$	
4	$(a, 7) \to (a, 6)$	$a_{-3} \to a_{-2} \in R_i$		$(1, I_3 I, 1')$ $I \to I_4 \in R_2$	$(i, g^{(i)} g_1^{(i)}, 2')$ $g_1^{(i)} \to \# \in R_i$	$(1, ee_1, 1')$ $(2, e1)$	
5	$(a, 6) \to (a, 5)$	$a_{-2} \to a_{-1} \in R_i$		$(2, I_4, 1)$ $I \to F \in R_{1'}$		$(1, ee_1, 1')$	

Simulation Synchronization Table

$F = f^{(1)} f_0^{(1)} \cdots f^{(m)} f_0^{(m)}$, $G = g^{(1)} g_0^{(1)} \cdots g^{(m)} g_0^{(m)}$

Step	Evolve	Idle, not halting	Halting	Control	G	ee_0	$F = f^{(1)} f_0^{(1)} \cdots f^{(m)} f_0^{(m)}$
6	$(a, 5) \to (a, 4)$	$a_{-1} \to a_0 \in R_i$		$(1, I_3 I_4, 1')$			$(1', f^{(i)}, i)$ $(1', f_0^{(i)}, i)$
7	$(a, 4) \to (a, 3)$	$a_0 \to a_i \in R_i$		$I_4 \to (I_0, 2) \in R_{1'}$			$f_0^{(i)} \to f_1^{(i)} \in R_i$ $(i, a^{(i)} f^{(i)}, 2')$
8	$(a, 3) \to (a, 2)$	$(i, a^{(i)} a_1, i')$		$(1', (I_0, 2), 1)$			$(i, f^{(i)} f_1^{(i)}, 2')$ $f_1^{(i)} \to \# \in R_i$
9	$(a, 2) \to (a, 1)$	$a_1 \to a \in R_{i'}$		$(I_0, 2) \to (I_0, 1) \in R_1$			
10	$(a, 1) \to a$	(i', a, i)		$(I_0, 1) \to I_0 \in R_1$			
Output	a	a		I_0			

Transitional P Systems with Randomized Rule Right-hand Sides

Artiom Alhazov[1,2*], Rudolf Freund[3], and Sergiu Ivanov[4,5]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău, MD-2028, Moldova
artiom@math.md

[2] Key Laboratory of Image Information Processing
and Intelligent Control of Education Ministry of China,
School of Automation,
Huazhong University of Science and Technology,
Wuhan 430074, China

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

[4] LACL, Université Paris Est – Créteil Val de Marne
61, av. Général de Gaulle, 94010, Créteil, France
sergiu.ivanov@u-pec.fr

[5] TIMC-IMAG/DyCTiM, Faculty of Medicine of Grenoble,
5 avenue du Grand Sablon, 38700, La Tronche, France
sergiu.ivanov@univ-grenoble-alpes.fr

Abstract. P systems are a model of hierarchically compartmentalized multiset rewriting. We introduce a novel kind of P systems in which rules are dynamically constructed in each step by non-deterministic pairing of left-hand and right-hand sides. We define three variants of right-hand side randomization and compare each of them with the power of conventional P systems. It turns out that all three variants enable non-cooperative P systems to generate exponential (and thus non-semi-linear) number languages. We also give a binary normal form for one of the variants of P systems with randomized rule right-hand sides.

1 Introduction

Membrane computing is a research field originally founded by Gheorghe Păun in 1998, see [12]. Membrane systems (also known as P systems) are a model of computing based on the abstract notion of a membrane. Formally, a membrane is treated as a container delimiting a region; a region may contain objects which are

* The work is supported by National Natural Science Foundation of China (61320106005 and 61033003) and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012).

acted upon by the rewriting rules associated with the membranes. Quite often, the objects are plain symbols coming from a finite alphabet, but P systems operating on more complex objects (e.g., strings, arrays) are often considered, too [9]. A comprehensive overview of different flavors of membrane systems and their expressive power is given in the handbook which appeared in 2010, see [13]. For a state of the art snapshot of the domain, we refer the reader to the P systems website [16], as well as to the bulletin of the International Membrane Computing Society [15].

Dynamic evolution of the set of available rules has been considered from the very beginning of membrane computing. Already in 1999, generalized P systems were introduced in [8]; in these systems the membranes, alongside the objects, contain *operators* which act on these objects, while the P system itself acts on the operators, thereby modifying the transformations which will be carried out on the objects in the subsequent steps. Among further ideas on dynamic rules, one may list rule creation [4], activators [1], inhibiting/deinhibiting rules [7], and symport/antiport of rules [6]. One of the more recent developments in this direction are *polymorphic P systems* [2, 3, 11], in which rules are defined by pairs of membranes, whose contents may be modified by moving objects in or out.

We remark that the previous studies on dynamic rule sets either treated the rules as atomic entities (symport/antiport of rules, operators in generalized P systems), or allowed virtually unlimited possibilities of tampering with their shape (polymorphic P systems). In the present work, we propose a yet different approach which can be seen as an intermediate one.

In *P systems with randomized rule-right-hand sides* (or with randomized RHS, for short), the available left-hand sides and right-hand sides of rules are fixed, but the associations between them are *re-evaluated in every step*: a left-hand side may pick a right-hand side arbitrarily (randomly). In Section 3, we present three different formal definitions of this intuitive idea of randomized RHS:

1. rules *exchange* their RHS,
2. each rule randomly picks an RHS from a *common* collection of RHS, *shared* between the rules,
3. each rule randomly picks an RHS from a possible collection of *RHS associated with the rule itself.*

P systems with randomized RHS may have a real-world (possibly biological) application for representing systems in a hostile environment. The modifications such P systems effect on their rules may be used to represent perturbations caused by the environment (mutations), somewhat in the spirit of faulty Turing machines (e.g., see [5]).

In this article, we will focus on the expressive power of P systems with randomized RHS, as well as on comparing them to the classical model with or without cooperative rules. One of the central conclusions of the present work is that non-cooperative P systems with randomized RHS can generate *exponential* number languages, thus (partially) surpassing the power of conventional (transitional) P systems.

This paper is structured as follows. Section 2 recalls some preliminaries about multisets, strings, permutations, as well as conventional transitional P systems. Section 3 defines the three variants of RHS randomization. Section 4 discusses the computational power of the three variants of P systems with randomized RHS. Section 5 shows a binary normal form for one of the variants of P systems with randomized RHS. Finally, Section 6 summarizes the results of the article and gives some directions for future work.

2 Preliminaries

In this paper, the set of positive natural numbers $\{1, 2, \dots\}$ is denoted by $\mathbb{N}^+$, the set of natural numbers also containing 0, i.e., $\{0, 1, 2, \dots\}$, is denoted by $\mathbb{N}$. Given $k \in \mathbb{N}^+$, we will call the set $\mathbb{N}^+{}_k = \{x \in \mathbb{N}^+ \mid 1 \leq x \leq k\}$ an *initial segment* of $\mathbb{N}^+$.

An *alphabet* V is a finite set. The families of recursively enumerable, context-free, linear, and regular languages, and of languages generated by tabled Lindenmayer systems are denoted by RE, CF, LIN, REG, and $ET0L$, respectively. The families of sets of Parikh vectors as well as of sets of natural numbers (multiset languages over one-symbol alphabets) obtained from a language family F are denoted by PsF and NF, respectively.

For further introduction to the theory of formal languages and computability, we refer the reader to [13, 14].

2.1 Linear Sets over $\mathbb{N}$

A *linear* set over $\mathbb{N}$ generated by a set of vectors $A = \{\mathbf{a}_i \mid 1 \leq i \leq d\} \subset_{fin} \mathbb{N}^n$ (here $A \subset_{fin} B$ indicates that A is a finite subset of B) and an offset $\mathbf{a}_0 \in \mathbb{N}^n$ is defined as follows:

$$\langle A, \mathbf{a}_0 \rangle_{\mathbb{N}} = \left\{ \mathbf{a}_0 + \sum\nolimits_{i=1}^{d} k_i \mathbf{a}_i \,\middle|\, k_i \in \mathbb{N},\ 1 \leq i \leq d \right\}.$$

If the offset $\mathbf{a}_0$ is the zero vector $\mathbf{0}$, we call the corresponding linear set *homogeneous*; we also use the short notation $\langle A \rangle_{\mathbb{N}} = \langle A, \mathbf{0} \rangle_{\mathbb{N}}$.

We use the notation $\mathbb{N}^n LIN_{\mathbb{N}} = \{\langle A, \mathbf{a}_0 \rangle_{\mathbb{N}} \mid A \subset_{fin} \mathbb{N}^n,\ \mathbf{a}_0 \in \mathbb{N}^n\}$, to refer to the class of all linear sets of n-dimensional vectors over $\mathbb{N}$. Semi-linear sets are defined as finite unions of linear sets. We use the notation $\mathbb{N}^n SLIN_{\mathbb{N}}$ to refer to the classes of semi-linear sets of n-dimensional vectors. In case no restriction is imposed on the dimension, n is replaced by $*$. We may omit n if $n = 1$. A finite union of linear sets which only differ in the starting vectors is called *uniform* semilinear:

$$\mathbb{N}^n SLIN_{\mathbb{N}}^U = \left\{ \bigcup\nolimits_{\mathbf{b} \in B} \langle A, \mathbf{b} \rangle_{\mathbb{N}} \mid A \subset_{fin} \mathbb{N}^n,\ B \subset_{fin} \mathbb{N}^n \right\}$$

Let us denote such a set by $\langle A, B \rangle_{\mathbb{N}}$.

Note that a uniform semilinear set $\langle A, B\rangle_{\mathbb{N}}$ can be seen as a pairwise sum of the finite set B and the homogeneous linear set $\langle A\rangle_{\mathbb{N}}$:

$$\langle A, B\rangle_{\mathbb{N}} = \{\mathbf{a} + \mathbf{b} \mid \mathbf{a} \in \langle A\rangle_{\mathbb{N}}, \mathbf{b} \in B\}.$$

This observation immediately yields the conclusion that the sum of two uniform semilinear sets $\langle A_1, B_1\rangle_{\mathbb{N}}$ and $\langle A_2, B_2\rangle_{\mathbb{N}}$ is uniform semilinear as well and can be computed in the following way:

$$\langle A_1, B_1\rangle_{\mathbb{N}} + \langle A_2, B_2\rangle_{\mathbb{N}} = \{\mathbf{a} + \mathbf{b} \mid \mathbf{a} \in \langle A_1 \cup A_2\rangle_{\mathbb{N}}, \mathbf{b} \in B_1 + B_2\}.$$

As is folklore,

$$PsCF = PsLIN = PsREG = \mathbb{N}^* SLIN_{\mathbb{N}}.$$

2.2 Multisets

A *multiset* over V is any function $w : V \to \mathbb{N}$; $w(a)$ is the *multiplicity* of a in w. A multiset w is often represented by one of the strings containing exactly $w(a)$ copies of each symbol $a \in V$. The set of all multisets over the alphabet V is denoted by V°. By abusing string notation, the empty multiset is denoted by λ. The *projection* (restriction) of w over a sub-alphabet $V' \subseteq V$ is the multiset $w|_{V'}$ defined as follows:

$$w|_{V'}(a) = \begin{cases} w(a), & a \in V'; \\ 0, & a \in V \smallsetminus V'. \end{cases}$$

Example 1. The string aab can represent the multiset $w : \{a, b\} \to \mathbb{N}$ with $w(a) = 2$ and $w(b) = 1$. The projection $w|_{\{a\}} = w'$ is defined as $w'(a) = w(a) = 2$ and $w'(b) = 0$.

We will (ab)use the symbol $\in$ to denote the relation "is a member of" for multisets. Therefore, for a multiset w, $a \in w$ will stand for $w(a) > 0$.

2.3 Strings and Permutations

A (non-empty) *string* s over an alphabet V traditionally is defined as a finite ordered sequence of elements of V. Equivalently, we can define a string of length k as a function assigning symbols to positions: $s : \mathbb{N}^+{}_k \to V$. Thus, the string $s = aab$ can be equivalently defined as the function $s : \mathbb{N}^+{}_3 \to \{a, b\}$ with $s(1) = a$, $s(2) = a$, and $s(3) = b$. We will use the traditional notation $|s|$ to refer to the length of the string s (i.e., the size k of the initial segment $\mathbb{N}^+{}_k$ it is defined on). In addition, the size of the empty string λ is 0.

A string $s : \mathbb{N}^+{}_k \to V$ is not necessarily surjective (there may be symbols from V that do not appear in s). We will use the notation $set(s)$ to refer to the set of symbols appearing in s (the image of s):

$$set(s) = \left\{a \in V \mid a = s(i) \text{ for some } i \in \mathbb{N}^+{}_{|s|}\right\}.$$

Given a string $s : \mathbb{N}^+{}_k \to V$, a *prefix* of length $k' \leq k$ of s is the restriction of s to $\mathbb{N}^+{}_{k'} \subseteq \mathbb{N}^+{}_k$. For example, aa is a prefix of length 2 of the string aab. We will use the notation $\mathrm{pref}_{k'}(s)$ to denote the prefix of length k' of s.

Given a finite set A, a *permutation* of A is any bijection $\rho : A \to A$. Given a permutation $\sigma : \mathbb{N}^+{}_k \to \mathbb{N}^+{}_k$ and a string $s : \mathbb{N}^+{}_k \to V$ of length k, *applying σ to s* is defined as $\sigma(s) = s \circ \sigma$ (where $\circ$ is the function composition operator).

Example 2. Following the widespread tradition, we will write permutations in Cauchy's two-line notation. The permutation σ_{rev} of $\mathbb{N}^+{}_3$ which "reverses the order" of the numbers, can be written as follows:

$$\sigma_{rev} = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}.$$

Applying σ_{rev} to a string reverses it:

$$\sigma_{rev}(aab) = baa.$$

Any finite set B trivially can be represented by one of the strings listing all of its elements exactly once. All such strings are equivalent modulo permutations. Given a fixed enumeration $B = \{b_1, \dots, b_n\}$, we define the *canonical string representation* of B to be the string $\delta(B) = b_1 \dots b_n$.

2.4 Rule Sides

We consider arbitrary labeled multiset rules $r : u \to v$ over an alphabet V, where r is the rule label we attach for convenience, and u and v are strings over V representing multisets. As usual, the application of such a rule means replacing the multiset represented by u by the multiset represented by v.

For a given rule $r : u \to v$, we define the left-hand-side and the right-hand-side functions as follows:

$$lhs(u \to v) = lhs(r) = (u),$$
$$rhs(u \to v) = rhs(r) = (v).$$

Using the brackets (and), for a given string w, the notation (w) is used to describe the multiset represented by w. As usual, we will extend the notations for these functions lhs and rhs lifted to sets of rules: given a set of rules R, $lhs(R) = \{lhs(r) \mid r \in R\}$ and $rhs(R) = \{rhs(r) \mid r \in R\}$. Furthermore, for any *string* (finite ordered sequence) of rules $\rho : \mathbb{N}^+{}_k \to R$ we define the strings of left-hand sides $lhs(\rho) = lhs \circ \rho$ and of right-hand sides $rhs(\rho) = rhs \circ \rho$.

Example 3. Take $R = \{r_1 : aa \to ab, r_2 : cc \to cd\}$ and consider the string of rules $\rho = r_1 r_1 r_2$. Then $lhs(\rho) = (aa)(aa)(cc)$ and $rhs(\rho) = (ab)(ab)(cd)$. Thus, $lhs(\rho)$ and $rhs(\rho)$ can be considered as *strings of multisets*.

We will (ab)use the symbol $\to$ for *combining* two strings of multisets $\alpha, \beta : \mathbb{N}^+{}_k \to V^\circ$ of the same length k. The *string* $\alpha \to \beta$ will be defined as follows, for any $i \in \mathbb{N}^+{}_k$:

$$(\alpha \to \beta)(i) = \alpha(i) \to \beta(i).$$

Example 4. Consider the following two strings of multisets: $\alpha = (aa)(aa)(cc)$ and $\beta = (ab)(ab)(cd)$. $\alpha \to \beta$ is simply the string of rules that can be obtained by taking the multisets from α as left-hand sides and β as right-hand sides, in the given order: $\alpha \to \beta = (aa) \to (ab)(aa) \to (ab)(cc) \to (cd)$ (which exactly corresponds with ρ from Example 3).

2.5 Transitional P Systems

A *transitional P system* is a construct

$$\Pi = (O, T, \mu, w_1, \dots, w_n, R_1, \dots R_n, h_i, h_o),$$

where O is the alphabet of objects, $T \subseteq O$ is the alphabet of terminal objects, μ is the membrane structure injectively labeled by the numbers from $\{1, \dots, n\}$ and usually given by a sequence of correctly nested brackets, w_i are the multisets giving the initial contents of each membrane i ($1 \leq i \leq n$), R_i is the finite set of rules associated with membrane i ($1 \leq i \leq n$), and h_i and h_o are the labels of the input and the output membranes, respectively ($1 \leq h_i \leq n$, $1 \leq h_o \leq n$).

In the present work, we will mostly consider the *generative case*, in which Π will be used as a multiset language-generating device. We therefore will systematically omit specifying the input membrane h_i.

Quite often the rules associated with membranes are multiset rewriting rules (or special cases of such rules). Multiset rewriting rules have the form $u \to v$, with $u \in O^o \setminus \{\lambda\}$ and $v \in O^o$. If $|u| = 1$, the rule $u \to v$ is called *non-cooperative*; otherwise it is called *cooperative*. Rules may additionally be allowed to send symbols to the neighboring membranes. In this case, for rules in R_i, $v \in O \times Tar_i$, where Tar_i contains the targets *out* (corresponding to sending the symbol to the parent membrane), *here* (indicating that the symbol should be kept in membrane i), and in_h (indicating that the symbol should be sent into the child membrane h of membrane i). Note that all variants of the function rhs, as well as the operator $\to$ from the previous section can be naturally extended to rules having right-hand sides with target indications (from $O \times Tar_i$).

In P systems, rules are often applied in the maximally parallel way: in any derivation step, a non-extendable multiset of rules has to be applied. The rules are not allowed to consume the same instance of a symbol twice, which creates competition for objects and may lead to the P system choosing non-deterministically between the maximal collections of rules applicable in one step.

A computation of a P system is traditionally considered to be a sequence of configurations it can successively pass through, stopping at the halting configuration. A halting configuration is a configuration in which no rule can be applied any more, in any membrane. The result of a computation of a P system Π as defined above is the contents of the output membrane h_o projected over the terminal alphabet T.

Example 5. For readability, we will often prefer a graphical representation of P systems. For example, the P system $\Pi_1 = (\{a, b\}, \{b\}, [_1\ ]_1, a, R, 1)$ with the rule set $R = \{a \to aa, a \to b\}$ may be depicted as in Figure 1.

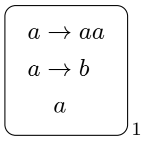


Fig. 1. The example P system Π_1

Due to maximal parallelism, at every step Π_1 may double some of the symbols a, while rewriting some other instances into b.

Note that, even though Π_1 might express the intention of generating the set of numbers of the powers of two, it will actually generate the whole of $\mathbb{N}^+$ (due to halting). Indeed, for any $n \in \mathbb{N}^+$, a^n can be generated in n steps by choosing to apply, in the first $n-1$ steps, $a \to aa$ to exactly one instance of a and $a \to b$ to all the other instances, and by applying $a \to b$ to every a in the last step (in fact, for $n > 1$, in each step except the last one, in which $a \to b$ is applied twice, both rules are applied exactly once, as exactly two symbols a are present, whereas all other symbols are copies of b).

While maximal parallelism and halting by inapplicability are staple ingredients, various other derivation modes and halting conditions have been considered for P systems, e.g., see [13].

We will use the notation $OP_n(coo)$ to denote the family of transitional P systems with at most n membranes, with cooperative rules. To denote the family of such P systems with *non-cooperative* rules, we replace *coo* by *ncoo*. To denote the family of languages of multisets generated by these P systems, we prepend Ps to the notation, and to denote the family of the generated number languages, we prepend N.

3 Transitional P Systems with Randomized RHS

In this section we consider three different variants of defining transitional P systems with randomized RHS. We immediately point out that, despite the common intuitive background, the details of the resulting semantics vary quite a lot.

3.1 Variant 1: Random RHS Exchange

In this variant of transitional P systems, rules randomly exchange right-hand sides at the beginning of every evolution step. This variant was the first to be conceived and is the closest to the classical definition.

A *transitional P system with random RHS exchange* is a construct

$$\Pi = (O, T, \mu, w_1, \dots, w_n, R_1, \dots R_n, h_o),$$

where the components of the tuple are defined as in the classical model (Section 2.5).

As different from conventional transitional P systems, Π does not apply the rules from R_i directly. Instead, for each membrane $1 \leq i \leq n$, we take the canonical representation of R_i, i.e., $\delta(R_i)$, and non-deterministically (randomly) choose a permutation $\sigma : \mathbb{N}^+{}_{|R_i|} \to \mathbb{N}^+{}_{|R_i|}$ to compute the canonical representation of R_i^σ from $\delta(R_i)$ as follows:

$$\delta(R_i^\sigma) = lhs(\delta(R_i)) \to \sigma(rhs(\delta(R_i))).$$

We now extract the set of rules $R_i^\sigma = set(\delta(R_i^\sigma))$ described by the string $\delta(R_i^\sigma)$ as constructed above. Π will then apply the rules from R_i^σ according to the usual maximally parallel semantics in membrane i.

In other words, Π *non-deterministically permutes* the right-hand sides of rules in each membrane i, and then applies the obtained rules according to the maximally parallel semantics.

Note that we first have to transform the set R_i into its canonical string representation $\delta(R_i)$ in order to be able to obtain a correct representation of the $|R_i|$ rules and from that a correct representation of the $|R_i|$ rules in R_i^σ, even if the number of different left-hand sides and/or different right-hand sides of rules does not equal $|R_i|$.

Example 6. Consider the P system $\Pi_2 = (\{a, b\}, \{b\}, [_1\ ]_1, a, R, 1)$ with the rule set $R = \{a \to aa, c \to b\}$. Π_2 is graphically represented in Figure 2.

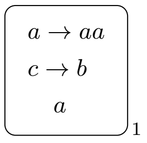


Fig. 2. The P system Π_2 with random RHS exchange generating the number language $\{2^n \mid n \in \mathbb{N}\}$.

The number language generated by Π_2 (the set of numbers of instances of b that may appear in the skin after Π_2 has halted) is exactly $\{2^n \mid n \in \mathbb{N}^+\}$. Indeed, while Π_2 applies the identity permutation on the right-hand sides, $a \to aa$ will double the number of symbols a, while the rule $c \to b$ will never be applicable. When Π_2 exchanges the right-hand sides of the rules, the rule $a \to b$ will rewrite every symbol a into a symbol b. After this has happened, no rule will ever be applicable any more and Π_2 will halt with 2^n symbols b in the skin, where $n + 1$ is the number of computation steps taken.

We will use the notation

$$OP_n(rhsExchange, coo)$$

to denote the family of transitional P systems with random RHS exchange, with at most n membranes, with cooperative rules. To denote the family of such P systems with *non-cooperative* rules, we replace *coo* by *ncoo*. To denote the family of languages of multisets generated by these P systems, we prepend Ps to the notation, and to denote the family of the generated number languages, we prepend N.

3.2 Variant 2: Randomized Pools of RHS

In this variant of transitional P systems, every membrane has some fixed left-hand sides and a *pool* of available right-hand sides to build rules from. An RHS from the pool can only be used once.

A *transitional P system with randomized pools of RHS* is a construct

$$\Pi = (O, T, \mu, w_1, \dots, w_n, H_1, \dots H_n, h_o),$$

where H_i defines the left- and right-hand sides available in membrane i and the other components of the tuple are defined as in the classical model (Section 2.5).

For $1 \leq i \leq n$, $H_i = (l_i, r_i)$ is a pair of strings of multisets over O. The string r_i may contain target indications (i.e., be a string of multisets over $O \times Tar_i$). The strings l_i and r_i are not necessarily of the same length. The length of the shortest of the two strings l_i and r_i is denoted by

$$k_i = \min(|l_i|, |r_i|).$$

At the beginning of every computation step in Π, for every membrane i, we construct the set of rules it will apply in the following way:

1. non-deterministically choose two (random) permutations

$$\sigma_l : \mathbb{N}^+{}_{|l_i|} \to \mathbb{N}^+{}_{|l_i|}, \quad \sigma_r : \mathbb{N}^+{}_{|r_i|} \to \mathbb{N}^+{}_{|r_i|};$$

2. take the first k_i elements out of $\sigma_l(l_i)$ and $\sigma_r(r_i)$:

$$l'_i = \mathrm{pref}_{k_i}(\sigma_l(l_i)), \quad r'_i = \mathrm{pref}_{k_i}(\sigma_r(r_i));$$

3. construct the set of rules R_i to be applied in membrane i by combining the left- and right-hand sides from l'_i and r'_i:

$$R_i = set(l'_i \to r'_i).$$

In step (3), we combine the strings l'_i and r'_i using the operator $\to$ defined in Subsection 2.4 and then apply the operator *set* to obtain the corresponding set of rules from the string representation.

After having constructed the set R_i for each membrane i, Π will proceed to applying the obtained rules according to the usual maximally parallel semantics.

When computing the strings l'_i and r'_i, we apply *two* different permutations σ_l and σ_r to l_i and r_i, in order to ensure fairness for the participation of left-hand

and right-hand sides when $|l_i| \neq |r_i|$. For example, if we only permuted r_i in the case in which $|l_i| > |r_i|$, the left-hand sides located at positions $k > |r_i|$ in l_i would never be used.

We do not explicitly prohibit repetitions in l_i or in r_i, but we avoid repeated rules by constructing R_i using the *set* function.

Example 7. Consider the following P system with randomized pools of RHS: $\Pi_3 = (\{a, b\}, \{b\}, [_1\ ]_1, a, H, 1)$, with $H = \big((a), (aa)(b)\big)$; (a) stands for the multiset containing an instance of a, while $(aa)(b)$ is the string denoting the two multisets (aa) and (b). The graphical representation of Π_3 is given in Figure 3.

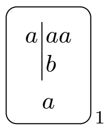


Fig. 3. The P system Π_3 with randomized pools of RHS generating the number language $\{2^n \mid n \in \mathbb{N}\}$.

The pair $H = (l, r)$ of strings of multisets is represented by listing the multisets of l and r in two columns and by drawing a vertical line between the two columns.

Π_3 follows exactly the same pattern as Π_2 from Example 6: while the identity permutation is applied to r, Π_3 keeps doubling the symbols a in the skin. Once the multisets (aa) and (b) are permuted in r, and thus the rule $a \to b$ is formed, all symbols a are rewritten into symbols b in one step and Π_3 must halt. Note that randomly taking the right-hand sides from a given pool avoids having the extra dummy rule $c \to b$ in Π_2.

We will use the notation

$$OP_n(\mathit{rhsPools}, \mathit{coo})$$

to denote the family of transitional P systems with randomized pools of RHS, with at most n membranes, with cooperative rules. To denote the family of such P systems with *non-cooperative* rules, we replace *coo* by *ncoo*. To denote the family of languages of multisets generated by these P systems, we prepend Ps to the notation, and to denote the family of the generated number languages, we prepend N.

3.3 Variant 3: Individual Randomized RHS

In this variant of transitional P systems, each rule is constructed from a left-hand side and a set of possible right-hand sides.

A *transitional P system with individual randomized RHS* is a construct

$$\Pi = (O, T, \mu, w_1, \dots, w_n, P_1, \dots P_n, h_o),$$

where P_i is the set of *productions* associated with the membrane i and the other components of the tuple are defined as in the classical model (Section 2.5).

A production is a pair $u \to R$, where $u \in O^\circ$ is the left-hand side and $R \subseteq O^\circ$ is a finite set of right-hand sides. The right-hand sides in R may have target indications, i.e., for a production in membrane i, we may consider $R \subseteq (O \times Tar_i)^\circ$. At the beginning of each computation step, for every membrane i, for each production $u \to R \in R_i$, Π will non-deterministically (randomly) pick a right-hand side v from R and will construct the rule $u \to v$ (this happens once per production). Π will then apply the rules thus constructed according to the maximally parallel semantics.

Example 8. Generating the language of the powers of two is the easiest compared with Variants 1 and 2. Indeed, consider the P system with individual randomized RHS $\Pi_4 = (\{a, b\}, \{b\}, [_1\ ]_1, a, P, 1)$ with only one production: $P = \{a \to \{aa, b\})\}$. Its graphical representation is given in Figure 4.

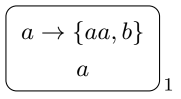


Fig. 4. The P system Π_4 with individual randomized RHS generating the number language $\{2^n \mid n \in \mathbb{N}\}$.

Π_4 works exactly like Π_2 and Π_3 from Examples 6 and 7: it doubles the number of symbols a and halts by rewriting them to b in the last step.

We will use the notation

$$OP_n(\mathit{rndRhs}, \mathit{coo})$$

to denote the family of transitional P systems with individual randomized RHS, with at most n membranes, with cooperative rules. To denote the family of such P systems with *non-cooperative* rules, we replace *coo* by *ncoo*. To denote the family of languages of multisets generated by these P systems, we prepend Ps to the notation, and to denote the family of the generated number languages, we prepend N.

We will sometimes want to set an upper bound k on the number of right-hand sides per production. To refer to the family of P systems with individual randomized RHS with such an upper bound, we will replace *rndRhs* by rndRhs^k in the notation above.

3.4 Halting with Randomized RHS

The conventional (total) halting condition for P systems can be naturally lifted to randomized RHS: a P system Π with randomized RHS (Variant 1, 2, or 3) halts on a configuration C if, however it permutes rule right-hand sides in Variant 1, or however it builds rules out of the available rule sides in Variants 2 and 3, no rule can be applied in C, in any membrane.

Note that, for Variants 1 and 3, the permutations chosen do *not* affect the applicability of rules, because applicability only depends on left-hand sides, which are always the same in any membrane. The situation is different for Variant 2, because the number of available left-hand sides in a membrane of Π may be bigger than the number of available right-hand sides. Therefore, if Π is a P system with randomized pools of RHS, the way rule sides are permuted may affect the number of rules applicable in a given configuration. This is why, for Π to halt on C, we require no rule to be applicable for any permutation.

In this paper, we will mainly consider P systems with randomized pools of RHS in which, in every membrane, there are at least as many right-hand sides as there are left-hand sides. To refer to P systems with this restriction, we will use the notation *rhsPools*$'$. In these systems, the problem with the applicability of rules as described above can be avoided.

3.5 Equivalence Between Variants 1 and 2

Before discussing the computational power of the P systems with randomized RHS in general, we will briefly point out a strong relationship between P systems with random RHS exchange and P systems with randomized pools of RHS, *with* the restriction that every membrane contains at least as many right-hand sides as it has left-hand sides, i.e., for P systems with randomized RHS of type *rhsPools*$'$.

Theorem 1. *For any $k \in \{coo, ncoo\}$, the following holds:*

$$PsOP_n(rhsExchange, k) = PsOP_n(rhsPools', k).$$

Proof. Any membrane with random RHS exchange trivially can be transformed into a membrane with randomized pools of RHS by listing the left-hand sides of the rules in the pool of LHS and the right-hand sides of the rules in the pool of RHS.

Conversely, consider a membrane i with randomized pools of RHS, with the string l_i of LHS and the string r_i of RHS, $|l_i| \leq |r_i|$. We can transform it into a membrane with random RHS exchange as follows. For every LHS u from l_i, pick (and remove) an RHS v from r_i, and construct the rule $u \to v$. According to our supposition, we will exhaust the LHS before (or at the same time as) the RHS. For every RHS v' which is left, we add a new (dummy) symbol z' to the alphabet, and add the rule $z' \to v'$. Since the symbol z' is new and does not appear in any RHS, it will never be produced and the rule $z' \to v'$ will essentially serve as a stash for the RHS v'. □

3.6 Flattening

The folklore flattening construction (see [13] for several examples as well as [10] for a general construction) is quite directly applicable to P systems with individual randomized RHS.

Proposition 1 (flattening). *For any $k \in \{coo, ncoo\}$, the following is true:*

$$PsOP_1(rndRhs, k) = PsOP_n(rndRhs, k).$$

Proof (sketch). Since in the case of individual randomized RHS, randomization has per rule granularity (whereas in the other two variants randomization occurs at the level of membranes), we can simulate multiple membranes by attaching membrane labels to symbols. For example, a production $ab \to \{cd, f\}$ in membrane h becomes $a_h b_h \to \{c_h d_h, f_h\}$, while the send-in production $a \to \{(b, in_i), (b, in_j)\}$ becomes $a_h \to \{b_i, b_j\}$. □

On the other hand, for Variants 1 and 2 similar results cannot be proved in such a way, a situation which happens very seldom in the area of P systems, especially in the case of variants of the standard model. Yet intuitively, it is easy to understand why this happens, as in both Variants 1 and 2 the right-hand sides in just one membrane can randomly be chosen for any left-hand side, whereas different membranes can separate the possible combinations of left-hand sides and right-hand sides of rules. A formal proof showing that flattening is impossible for the types *rhsExchange* and *rhsPools′* will be given in the succeeding section by constructing a suitable example.

4 Computational Power of Randomized RHS

In this section, we look into the computational power of the three different versions of P systems with randomized right-hand sides. We first shortly consider the case of cooperative rules and then focus on the case of non-cooperative rules.

4.1 Cooperative Rules

The following result concerning the relationship between P systems with individual randomized RHS and conventional transitional P systems holds for both cooperative and non-cooperative rules:

Proposition 2. *For $\alpha \in \{ncoo, coo\}$, $PsOP_n(rndRhs, \alpha) \supseteq PsOP_n(\alpha)$.*

Proof. Any conventional transitional P system can be trivially seen as a P system with individual randomized RHS in which every production has exactly one right-hand side. □

Now, the computational completeness of *cooperative* transitional P systems trivially implies the computational completeness of P systems with individual randomized RHS.

Corollary 1. *$PsOP_n(rndRhs, coo) = PsRE$.*

4.2 Non-cooperative Rules

First we mention an upper bound for the families $PsOP_n(\rho, ncoo)$, for any variant $\rho \in \{rhsExchange, rhsPools', rndRhs\}$:

Proposition 3. *For* $\rho \in \{rhsExchange, rhsPools', rndRhs\}$*,*

$$PsOP_n(\rho, ncoo) \subseteq PsET0L.$$

Proof. No matter how the rule sets are constructed in the three different variants, we always get a finite set of different sets of rules—*tables*—corresponding to tables in $ET0L$-systems, which can also mimic the contents of different membranes in the usual way by using symbols marked with the corresponding membrane label. □

Next we show one of the central results of this paper: randomized rule right-hand sides allow for generating *non-semilinear languages* already in the non-cooperative case.

Theorem 2. *The following is true for* $\rho \in \{rhsExchange, rhsPools', rndRhs\}$*:*

$$\{2^m \mid m \in \mathbb{N}\} \in NOP_n(\rho, ncoo) \setminus NOP_n(ncoo).$$

Proof. The statement follows (for $n \geq 1$) from the constructions given in Examples 6, 7, and 8 and from the well-known fact that non-cooperative P systems operating under the total halting condition cannot generate non-semilinear number languages (for example, see [13]). □

This result is somewhat surprising at a first glance, but becomes less so when one remarks that the constructions from all three examples only effectively use *one rule* to do the multiplication, which is non-deterministically changed to a "halting" rule. Since there is only one rule acting at any time, randomized right-hand sides allow for clearly delimiting different *derivation phases*.

It turns out that this approach of synchronization by randomization can be exploited to generate even more complex non-semilinear languages.

Theorem 3. *Given a fixed subset of natural factors* $\{f_1, \dots, f_k\} \subseteq \mathbb{N}$*, the following is true for any* $\rho \in \{rhsExchange, rhsPools', rndRhs\}$*:*

$$L = \{f_1^{n_1} \cdot \dots \cdot f_k^{n_k} \mid n_1, \dots, n_k \in \mathbb{N}\} \in NOP_1(\rho, ncoo).$$

Proof. First consider the P system with randomized pools of RHS $\Pi_5 = (\{a, b\}, \{b\}, [_1\ ]_1, a, H, 1)$ with $H = (l, r)$, $l = (a)$ and $r = \left(a^{f_1}\right) \dots \left(a^{f_k}\right) \left(b\right)$. This P system is graphically represented in Figure 5.

Similarly to the P systems from Examples 6, 7, and 8, Π_5 halts by choosing to pick the right-hand side b and constructing the rule $a \to b$. If Π_5 picks a different right-hand side, it will multiply the contents of the skin membrane (membrane 1) by one of the factors f_i, $1 \leq i \leq k$. This proves that $L \in NOP_1(rhsPools', ncoo)$,

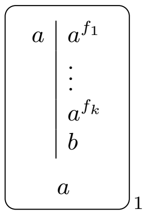


Fig. 5. The P system Π_5 with randomized pools of RHS generating the number language $\{f_1^{n_1} \cdot \ldots \cdot f_k^{n_k} \mid n_1, \ldots, n_k \in \mathbb{N}\}$.

and, according to Theorem 1, $L \in NOP_1(rhsExchange, ncoo)$ as well: take the P system with the rules $\{a \to a^{f_1}, z_2 \to a^{f_2}, \ldots, z_k \to a^{f_k}, z_{k+1} \to b\}$ (the rules with z_j in their left-hand sides are dummy rules).

To show that $L \in NOP_1(rndRhs, ncoo)$, just construct a P system with the only production $a \to \{a^{f_1}, \ldots, a^{f_k}, b\}$. □

Therefore, randomizing the right-hand sides of rules in non-cooperative transitional P systems allows for generating non-semilinear languages which cannot be generated without randomization. A natural question to ask is whether randomizing the RHS leads to a *strict increase* in the computational power. The answer is trivially positive for P systems with individual randomized RHS (Variant 3).

Proposition 4. *$PsOP_n(rndRhs, ncoo) \supsetneq PsOP_n(ncoo)$.*

Proof. The inclusion follows from Proposition 2, as any conventional transitional P system can be trivially seen as a P system with individual randomized RHS in which every production has exactly one right-hand side. Theorem 3 proves the strictness of the inclusion. □

On the other hand, the other two variants of randomizing right-hand sides—random RHS exchange (Variant 1) and randomized pools of RHS (Variant 2)—actually *prevent* one-membrane P systems with non-cooperative rules from generating some semilinear languages, which result also shows that flattening is not possible for these two variants.

In what follows, we will use the expression “only one rule is applied” to refer to the fact that only one given rule $u \to v$ is applied in a certain configuration, possibly in multiple copies. Dually, by saying “at least two rules are applied”, we mean that at least two different rules, $u \to v$ and $u' \to v'$, are applied, possibly in multiple copies each.

Theorem 4. *For $\rho \in \{rhsExchange, rhsPools'\}$, the following holds:*

$$L_{ab} = \{a^n \mid n \in \mathbb{N}\} \cup \{b^n \mid n \in \mathbb{N}\} \notin PsOP_1(\rho, ncoo).$$

Proof. Consider a P system Π with randomized RHS of the variant given by ρ and with non-cooperative rules. We immediately remark that no left-hand side in Π may be a or b, because in this case Π will never be able to halt with its only (skin) membrane containing either the multiset a^n or b^n. Furthermore, any RHS of Π contains combinations of symbols a, b, or LHS symbols. Indeed, if an RHS contained a symbol not belonging to these three classes, instances of this symbol would pollute the halting configuration. Finally, Π contains no RHS v such that $a \in v$ and $b \in v$. If Π did contain such an RHS, then any computation could be hijacked to produce a mixture of symbols a and b.

With these remarks in mind, the statement of the theorem follows from the contradicting Lemmas 1 and 2, which are shown immediately after this proof.

□

Lemma 1. *Take a $\Pi \in OP_1(\rho, ncoo)$, $\rho \in \{rhsExchange, rhsPools'\}$, such that it generates the number language $Ps(\Pi) = L_{ab}$. Then it must have a computation in which more than one rule is applied (two different left-hand sides are employed) in at least one step.*

Proof. Suppose that Π applies exactly one rule in every step of every computation. We make the following two remarks:

1. Since the words in L_{ab} are of unbounded length, Π must have an LHS t and an RHS v such that $t \in v$, otherwise all computations of Π would have one step and would only produce words of bounded length.
2. Every such RHS v must contain at most one kind of LHS, i.e., if t_1 and t_2 are two LHS of Π then $t_1 \in v$ and $t_2 \in v$ implies $t_1 = t_2$. If this were not the case, after using v, Π would *have* to apply two different rules (assuming that Π has at least as many RHS as LHS).

According to these observations, as well as to those from the proof of Theorem 4, any RHS v of Π is the of the form $v = \alpha\beta$, where $\alpha \in \{a^k, b^k \mid k \in \mathbb{N}\}$, $\beta \in \{t^k \mid k \in \mathbb{N}\}$, and t is an LHS of Π. Note that both α and β may be empty. According to observation (1), Π must have at least an RHS for which $\beta \neq \lambda$ and there exists such an RHS which must be applied an unbounded number of times.

In what follows, we will separately treat the cases in which Π contains or does not contain mixed RHS, i.e., RHS for which both $\alpha \neq \lambda$ and $\beta \neq \lambda$.

No mixed RHS: Suppose that any RHS of Π which contains a left-hand side is of the form t_2^k. Then, according to our previous observations on the possible forms of the RHS of Π, all RHS containing a are of the form a^i and all RHS containing b are of the form b^j. According to the remarks from the proof of Theorem 4, a and b must not be LHS of Π. Therefore, in any computation of Π, all of a's and b's are produced in the last step. But then, the number of terminal symbols Π produces in a computation can be calculated as a product of the sizes of the RHS of the rules it has applied, which implies that there exists such a $p \in \mathbb{N}$ such that $a^p \notin Ps(\Pi)$ and therefore $Ps(\Pi) \neq L_{ab}$. (p may be picked to be the smallest prime number greater than the length of the longest RHS of Π.)

Mixed RHS: It follows from the previous paragraph that, in order to generate the number language L_{ab}, Π should contain and apply at least one RHS of the form $a^i t_1^{k_1}$ and at least one RHS of the form $b^j t_2^{k_2}$. Take a computation C of Π producing a and applying the rule $t \to a^i t_1^{k_1}$ at a certain step. Instead of this rule, apply $t \to b^j t_2^{k_2}$, and, in the following step, the rule $t_2 \to a^i t_1^{k_1}$. (We can do so because Π is allowed to pick any permutation of RHS.) Now, Π may continue applying the same rules as in C and eventually halt with a configuration containing *both* a and b. This implies that $Ps(\Pi) \neq L_{ab}$.

It follows from our reasoning that, if Π applies exactly one rule in any step of any computation, it cannot produce L_{ab}, which proves the lemma. $\square$

Lemma 2. *Take a $\Pi \in OP_1(\rho, ncoo)$, $\rho \in \{rhsExchange, rhsPools'\}$, such that it generates the number language $Ps(\Pi) = L_{ab}$. Then, in every computation of Π, exactly one rule is applied (one left-hand side is employed) in every step.*

Proof. Suppose that, in every computation of Π, there exists a step at which at least two different rules are applied. This immediately implies that Π has no RHS of the form a^i or b^j, for $i, j \geq 0$. Indeed, consider a computation producing the multiset a^n and a step in it at which more than one rule is applied. Then Π can replace one of the RHS introduced into the system at this step by b^j and thus end up with a mix of a's and b's in the halting configuration. Therefore, all RHS of Π containing a have the form $a^i v_a$ and all RHS containing b have the form $b^j v_b$, where v_a and v_b are non-empty multisets which only contain LHS symbols (which are neither a nor b).

Now, consider a computation C_a of Π halting on the multiset a^n, and take the *last* step s_a at which at least two different rules are applied. We will consider three different cases, based on whether a and an LHS t appear in the configurations of C_a *after* step s_a.

Both a and t are present: Suppose both a and an LHS t are present at step $s_a + 1$ in computation C_a. Then t is the only LHS present, because, by our hypothesis, only one rule is applied (maybe in multiple instances) at step $s_a + 1$. In this case, replace the rule applied at step $s_a + 1$ in C_a by $t \to b^j v_b$, where $b^j v_b$ is a right-hand side of Π used in a computation C_b producing b's. From step $s_a + 2$ on in the modified computation, just apply the same rules as applied to the symbols of v_b (and to those derived from v_b) in C_b. The modified computation will reach a halting configuration containing a mix of a's and b's.

Only a is present: Suppose only a is present at step $s_a + 1$ in computation C_a. Then all of the RHS used at step s_a are λ, because Π has no RHS of the form a^i. Then, replace one of these empty RHS by $b^j v_b$, where $b^j v_b$ is a right-hand side of Π used in a computation C_b producing b's. As before, just apply the same rules as in C_b in the modified computation to get a mix of a's and b's in the halting configuration.

No symbols a are present: Suppose now that there are no instances of a present at step $s_a + 1$ in computation C_a. Recall that Π has no RHS of the form a^i. Since we suppose that s_a is the last step at which at least two different rules are applied, this means that, in order to produce any a's in C_a, Π must have and use an RHS of the form $a^i t^k$. This RHS contains (multiple copies of) exactly one kind of LHS symbol: t.

Consider a computation C_b halting on the multiset b^n. We pick n sufficiently big to ensure that C_b uses at least two RHS containing b: $b^j v_b$ and $b^{j'} v'_b$ (possibly the same). Without losing generality, we may suppose that these two RHS are either used at the same step in C_b or that $b^{j'} v'_b$ is used at a later step than $b^j v_b$. Then, replace $b^{j'} v'_b$ by $a^i t^k$, pick one of the LHS symbols $t' \in v'_b$ and apply the same rules to t (and to the symbols derived from t) in the modified derivation as were applied to t' (and to the symbols derived from t') in C_b. The modified derivation will therefore contain a mix of a's and b's in the halting configuration.

It follows from our reasoning that, if in any derivation of Π there is a step at which at least two different rules are applied, then $Ps(\Pi) \neq L_{ab}$, which proves the lemma. □

The previous two lemmas are contradicting each other, which means that there exist no one-membrane P systems with random RHS exchange or with random pools of RHS which generate the union language $L_{ab} = \{a^n \mid n \in \mathbb{N}\} \cup \{b^n \mid n \in \mathbb{N}\}$ (this is the statement of Theorem 4). Together with Theorem 3, this leads us to the curious conclusion that one-membrane non-cooperative P systems with random RHS exchange or with randomized pools of RHS are *incomparable* in power to the conventional transitional P systems.

Corollary 2. *For $\rho \in \{rhsExchange, rhsPools'\}$, the following two statements are true:*

$$PsOP_1(\rho, ncoo) \setminus PsOP_1(ncoo) \neq \emptyset, \tag{1}$$

$$PsOP_1(ncoo) \setminus PsOP_1(\rho, ncoo) \neq \emptyset. \tag{2}$$

Proof. Statement (1) follows from Theorem 3. Statement (2) follows from Theorem 4. □

Theorem 4 also allows us to draw a negative conclusion as to the computational completeness of one-membrane non-cooperative P systems with random RHS exchange (Variant 1) and non-cooperative P systems with randomized pools of RHS (Variant 2).

Corollary 3. *For $\rho \in \{rhsExchange, rhsPools'\}$, the following is true:*

$$PsOP_1(\rho, ncoo) \subsetneq PsRE.$$

Note, however, that allowing multiple membranes strictly increases the expressive power of Variants 1 and 2 and allows easily generating the language L_{ab}, as shown by the following proposition.

Proposition 5. *For $\rho \in \{rhsExchange, rhsPools'\}$, the following holds:*

$$L_{ab} = \{a^n \mid n \in \mathbb{N}\} \cup \{b^n \mid n \in \mathbb{N}\} \in PsOP_3(\rho, ncoo).$$

Proof. Consider the following P system with randomised pools of RHS:

$$\Pi_6 = \big(\{p, a, b\}, \{a, b\}, [\,[\ ]_2 [\ ]_3\,]_1, p, \lambda, \lambda, H_1, H_2, H_3, 1\big),$$

where $H_i = (l_i, r_i)$ are given by the following:

$$\begin{aligned} l_1 &= (p), & r_1 &= \big((p, in_1)\big)\,\big((p, in_2)\big), \\ l_2 &= (p), & r_2 &= \big(p\,(a, out)\big)\,\big(\lambda\big), \\ l_3 &= (p), & r_3 &= \big(p\,(b, out)\big)\,\big(\lambda\big). \end{aligned}$$

This P system is graphically represented in Figure 6. Π_6 starts by sending p into either membrane 2 or 3, where p is either maintained, spawning off copies of a or b out into the skin membrane, or is rewritten into λ, thereby halting the system. Since the rules producing a's and b's are assigned to separate membranes, and since Π_6 only contains one copy of p at any moment, production of a's and b's cannot happen in the same evolution, which proves that $L_{ab} \in Ps(\Pi_6)$.

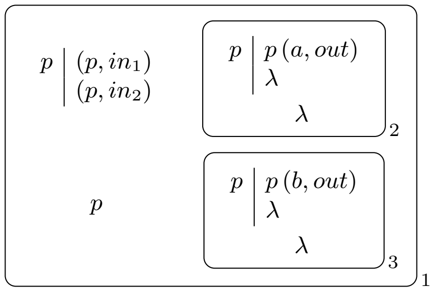


Fig. 6. The 3-membrane P system Π_6 with randomized pools of RHS generating the multiset language $L_{ab} = \{a^n \mid n \in \mathbb{N}\} \cup \{b^n \mid n \in \mathbb{N}\}$.

To complete the proof, we evoke Theorem 1 to show that there exists a P system with random RHS exchange (Variant 1) generating the same language L_{ab}.

This proposition allows us to draw a definitive conclusion about the impossibility of flattening for non-cooperative Variants 1 and 2, in contrast to Proposition 1 showing the opposite result for Variant 3.

Corollary 4. *For $\rho \in \{rhsExchange, rhsPools'\}$ and any $k \geq 2$, the following holds:*

$$PsOP_1(\rho, ncoo) \subsetneq PsOP_k(\rho, ncoo).$$

We conclude this section with two observations regarding the computational power of the Variants 1 and 2. We have seen that, with a single membrane and without cooperation, such P systems cannot generate all semilinear languages; yet it turns out they can generate all *linear* languages.

Theorem 5. *For $\rho \in \{rhsExchange, rhsPools'\}$, the following is true:*

$$\mathbb{N}^* LIN_{\mathbb{N}} \subseteq PsOP_1(\rho, ncoo).$$

Proof. Consider the arbitrary integer vectors $x, y_1, \ldots, y_n$ of the same dimension d and the linear set $A = \{x + ky_i \mid 1 \le i \le n, k \in \mathbb{N}\}$. We will now construct the P system $\Pi = (O, T, [\ ]_1, w_0, H, 1)$ with pools of randomized RHS in the following way:

- $O = \{a_1, \ldots, a_d, t\}$ contains a symbol per each dimension of the vectors, plus the special symbol t,
- $T = \{a_1, \ldots, a_d\}$ contains exactly the symbols representing the dimensions,
- $w_0 = w'_0 t$, such that the Parikh vector of w'_0 corresponds to the initial offset of A: $Ps(w'_0) = x$,
- $H = (l, r)$, with $l = (t)$ and $r = (w'_1 t) \ldots (w'_n t)(\lambda)$, such that $Ps(w'_i) = y_i$, $1 \le i \le n$.

At every step, Π either chooses one of the RHS $(w'_i t)$ which will enable it to reuse the left-hand side symbol t in the following step, or it constructs the rule $t \to \lambda$ which erases the only instance of t and halts the system. Thus, Π performs arbitrary additions of vectors y_i to the initial offset x, or, in other words, $Ps(\Pi) = A$. The fact that we can construct such a P system Π for any arbitrary linear set A proves the statement of the theorem. □

Even though non-cooperative P systems with random RHS exchange and with randomized pools of RHS cannot generate all unions of linear languages (Theorem 4), they can still generate some limited unions of exponential languages.

Theorem 6. *For $\rho \in \{rhsExchange, rhsPools'\}$, the following is true:*

$$L'_{ab} = \left\{a^{2^n} \mid n \in \mathbb{N}\right\} \cup \left\{b^{2^n} \mid n \in \mathbb{N}\right\} \in PsOP_1(\rho, ncoo).$$

Proof. A P system Π_7 generating the language L'_{ab} can be constructed as follows: $\Pi_7 = (\{a, b, t\}, \{a, b\}, [\ ]_1, t, H, 1)$, where $H = (l, r)$, $l = (t)$ and $r = (tt)(a)(b)$. A graphical representation of Π_7 is given in Figure 7.

Π_7 works by sequentially multiplying the number of symbols t by 2, until it decides to rewrite every instance of t to a or every instance of t to b. Therefore, $Ps(\Pi_7) = L'_{ab}$. According to Proposition 1, there also exists a P system with random RHS exchange generating L'_{ab}, which completes the proof. □

The construction from the previous proof can be clearly extended to any number of distinct terminal symbols and to any function of the number of steps

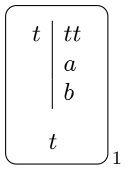


Fig. 7. The P system Π_7 with randomized pools of RHS generating the union language $L'_{ab} = \left\{a^{2^n} \mid n \in \mathbb{N}\right\} \cup \left\{b^{2^n} \mid n \in \mathbb{N}\right\}$

$f(n)$ given by a product of exponentials (like in Theorem 3). That is, one can construct a P systems with random RHS exchange or with randomized pools of RHS generating the union language $\left\{a_i^{f(n)} \;\middle|\; n \in \mathbb{N}, 1 \le i \le m\right\}$, for some fixed number m. Note, however, that we cannot use the same approach to generate unions of two different exponential functions. We conjecture that generating such unions is entirely impossible with Variants 1 and 2 of randomized RHS.

5 Variant 3: A Binary Normal Form

In this section we present a binary normal form for P systems with individual randomized RHS: we prove that, for any such P system, there exists an equivalent one in which every production has at most two right-hand sides.

We now introduce a (rather common) construction: symbols with finite timers attached to them. Given an alphabet O, we define the following two functions:

$$\begin{aligned} timers_o(t, O) &= \bigcup_{i=1}^{t} \{\langle a, i\rangle \mid a \in O\}, \\ timers_r(t) &= \{\langle a, i\rangle \to \langle a, i-1\rangle \mid 2 \le i \le t\} \\ &\cup \{\langle a, 1\rangle \to a \mid a \in O\}. \end{aligned}$$

Informally, $timers_o(t, O)$ attaches a t-valued timer to every symbol in O, while $timers_r(t)$ contains the rules making this timer work.

We also define the following function setting a timer to the value $t > 0$ for each symbol in a given string $a_1 \dots a_n$:

$$wait(t, a_1 \dots a_n) = \langle a_1, t\rangle \dots \langle a_n, t\rangle.$$

For $t = 0$, $wait$ is defined to be the identity function: $wait(0, a_1 \dots a_n) = a_1 \dots a_n$.

We can now show that, for any P system with individual randomized RHS there exists an equivalent one having at most two RHS per production.

Theorem 7 (normal form). *For any $\Pi \in OP_n(rndRhs, k)$, $k \in \{coo, ncoo\}$, there exists a $\Pi' \in OP_n(rndRhs^2, k)$ such that $Ps(\Pi') = Ps(\Pi)$.*

Proof. Consider the following P system with individual randomized RHS $\Pi = (O, T, \mu, w_1, \ldots, w_n, P_1, \ldots P_n, h_o)$ that has at least one production with more than two RHS. We will construct another P system with individual randomized RHS $\Pi' = (O', T, \mu, w_1, \ldots, w_n, P'_1, \ldots P'_n, h_o)$ such that $Ps(\Pi') = Ps(\Pi)$. The new alphabet will be defined as

$$O' = O \cup \mathit{timers}_o(t, O) \cup \{p_1, \ldots, p_t \mid p \in V_p\},$$

where $t + 2$ is the number of right-hand sides in the productions of Π having the most of them, and V_p is an alphabet containing a symbol for each of the individual productions of Π. (If there are two identical productions in Π which belong to two different membranes, V_p will contain one different symbol for each of these two productions.)

For every membrane $1 \leq i \leq n$, the new set of productions P'_i is constructed by applying the following procedure to every production $p \in P_i$:

– If p has the form $u \to \{v\}$, we add the production $u \to \{\mathit{wait}(t, v)\}$ to P'_i.
– If p has the form $u \to \{v_1, v_2\}$, we add $u \to \{\mathit{wait}(t, v_1), \mathit{wait}(t, v_2)\}$ to P'_i.
– If p has the form $u \to \{v_1, \ldots, v_k\}$, with $k \geq 3$, we add the following productions to P_i:

$$\begin{aligned} &\big\{u \to \{\mathit{wait}(t, v_1), p_1\}\big\} \\ \cup\, &\big\{p_j \to \{\mathit{wait}(t - j, v_{j+1}), p_{j+1}\} \mid 1 \leq j < k - 2\big\} \\ \cup\, &\big\{p_{k-2} \to \{\mathit{wait}(t - k + 2, v_{k-1}), \mathit{wait}(t - k + 2, v_k)\}\big\}. \end{aligned}$$

These productions are graphically represented in Figure 8, in which arrows go from LHS to the associated RHS.

$$\begin{tikzcd}
 & \mathit{wait}(t, v_1) & & & \mathit{wait}(t - j, v_{j+1}) & & & \mathit{wait}(t - k + 2, v_{k-1}) \\
u \arrow[r] \arrow[ur] & p_1 \arrow[r] & \ldots \arrow[r] & p_j \arrow[r] \arrow[ur] & p_{j+1} \arrow[r] & \ldots \arrow[r] & p_{k-2} \arrow[r] \arrow[ur] & \mathit{wait}(t - k + 2, v_k)
\end{tikzcd}$$

Fig. 8. Timers allow sequential choice between any number of right-hand sides.

Finally we add the rules from $\mathit{timers}_r(t)$, treated as one-RHS production, to every P'_i.

Instead of directly choosing between the right hand-sides of a production $p : u \to \{v_1, \ldots, v_k\}$ in one step, Π' chooses between v_1 and delaying the choice to the next step, by producing p_1. This choice between settling on an RHS or continuing the enumeration in the next step may be kept on until $k - 2$ RHS have been discarded. If p_{k-2} is reached, Π' must choose one of the two remaining RHS.

Thus, Π' evolves in "macro-steps", each consisting of exactly t steps. In the first step of a "macro-step", Π' acts on the symbols from O, producing some symbols with timers and delaying some of the choices by producing symbols p_j. All symbols with timers wait exactly until the t-th step of the "macro-step" to turn into the corresponding clean versions from O. Since $t + 2$ is the number of RHS in the biggest production of Π, Π' has the time to enumerate all of the RHS of this production.

Since every delayed choice of Π' is uniquely identified by a production-specific symbol p_j, and since only the productions from $timers_r(t)$ can act upon the symbols with timers in Π', the simulations of two different productions of Π cannot interfere. This concludes the proof of the normal form. □

6 Conclusions and Open Problems

In this article, we introduced and partially studied P systems with randomized rule right-hand sides. This is a model of P systems with dynamic rules, in which the matching between left-hand and right-hand sides is non-deterministically changed during the evolution. In each step, such P systems first construct the rules from the available rule sides and then apply them, in a maximally parallel way.

We defined three different randomization semantics: random RHS exchange (Variant 1), randomized pools of RHS (Variant 2), and individual randomized RHS (Variant 3). We studied the computational power of the three variants and showed that Variant 3 is quite different in power from Variants 1 and 2. Indeed, P systems with individual randomized RHS (Variant 3) appear as a strict extension of conventional transitional P systems, while random RHS exchange (Variant 1) and randomized pools of RHS (Variant 2) seem to increase the power when only one LHS is used, but to decrease the power when more LHS are present. Finally, we gave a binary normal form for P systems with individual randomized RHS (Variant 3).

The present work leaves open quite a number of questions. We list the ones appearing important to us, in no particular order.

Full power of Variants 1 and 2: Are cooperative, multi-membrane P systems with random RHS exchange (Variant 1) or with randomized pools of RHS (Variant 2) computationally complete? If not, what would be the upper bound on their power? In this article, we showed that applying these two randomization semantics to the non-cooperative, one-membrane case, yields a family of multiset languages incomparable with the family of semi-linear vector sets. How much more can be achieved with cooperativity? We conjecture that, even with LHS containing more than one symbol, Variants 1 and 2 will *not* be computationally complete. However, we expect that considering systems with multiple membranes may actually bring a substantial boost in computational power, because, in both Variants 1 and 2, randomization happens over each single membrane, meaning that one might use a rich membrane structure to finely control its effects.

Compare the variants: How do the three variants of RHS randomization compare among one another when applied to non-cooperative rules? We saw that, in all three cases, exponential number languages can be generated. We also saw that individual randomized RHS (Variant 3) produce a strict superset of the semi-linear languages (Proposition 4). Does it imply that Variant 3 is strictly more powerful than Variants 1 and 2? We conjecture a positive answer to this question.

Excess of LHS: In the case of P systems with randomized pools of RHS (Variant 2), what is the consequence of having *more LHS* available in a membrane than there are RHS? The results in this paper concern a "restricted" version of Variant 2, in which we require that LHS are never in excess. How strong is this restriction? Our conjecture is that allowing an excess of LHS does not increase the computational power.

Applications to vulnerable systems: As noted in the introduction to the present work, randomized RHS can be seen as a representation of systems mutating in a toxic environment. However, we did not give any concrete examples. It would be interestng to look up any such concrete cases and to evaluate the relevance of this unconventional modeling approach.

References

1. Artiom Alhazov. A note on P systems with activators. In Gheorghe Păun, Agustín Riscos-Núñez, Alvaro Romero-Jiménez, and Fernando Sancho-Caparrini, editors, *Second Brainstorming Week on Membrane Computing, Sevilla, Spain, February 2-7 2004*, pages 16–19, 2004.
2. Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, and Marion Oswald. Observations on P systems with states. In Marian Gheorghe, Ion Petre, Mario J. Pérez-Jiménez, Grzegorz Rozenberg, and Arto Salomaa, editors, *Multidisciplinary Creativity. Hommage to Gheorghe Păun on His 65th Birthday*. Spandugino, 2015.
3. Artiom Alhazov, Sergiu Ivanov, and Yurii Rogozhin. Polymorphic P systems. In Marian Gheorghe, Thomas Hinze, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 6501 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2011.
4. Fernando Arroyo, Angel V. Baranda, Juan Castellanos, and Gheorghe Păun. Membrane computing: The power of (rule) creation. *Journal of Universal Computer Science*, 8:369–381, 2002.
5. Ilir Çapuni and Péter Gács. A Turing machine resisting isolated bursts of faults. *CoRR*, abs/1203.1335, 2012.
6. Matteo Cavaliere and Daniela Genova. P systems with symport/antiport of rules. In Gheorghe Păun, Agustín Riscos-Núñez, Alvaro Romero-Jiménez, and Fernando Sancho-Caparrini, editors, *Second Brainstorming Week on Membrane Computing, Sevilla, Spain, February 2–7 2004*, pages 102–116, 2004.
7. Matteo Cavaliere, Mihai Ionescu, and Tseren-Onolt Ishdorj. Inhibiting/de-inhibiting rules in P systems. In *Pre-proceedings of the Fifth Workshop on Membrane Computing (WMC5), Milano, Italy, June 2004*, pages 174–183, 2004.
8. Rudolf Freund. Generalized P-Systems. In Gabriel Ciobanu and Gheorghe Păun, editors, *Fundamentals of Computation Theory, 12th International Symposium, FCT '99, Iaşi, Romania, August 30–September 3,1999, Proceedings*, volume 1684 of *Lecture Notes in Computer Science*, pages 281–292. Springer, 1999.

9. Rudolf Freund. *P Systems Working in the Sequential Mode on Arrays and Strings*, pages 188–199. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
10. Rudolf Freund, Alberto Leporati, Giancarlo Mauri, Antonio E. Porreca, Sergey Verlan, and 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.
11. Sergiu Ivanov. Polymorphic P systems with non-cooperative rules and no ingredients. 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 258–273. Springer, 2014.
12. Gheorghe Păun. Computing with Membranes. *Journal of Computer and System Sciences*, 61:108–143, 1998.
13. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *The Oxford Handbook of Membrane Computing.* Oxford University Press, Inc., New York, NY, USA, 2010.
14. Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages, 3 volumes.* Springer, New York, NY, USA, 1997.
15. Bulletin of the International Membrane Computing Society (IMCS). http://membranecomputing.net/IMCSBulletin/index.php.
16. The P Systems Website. http://ppage.psystems.eu/.

On Reversibility in Reaction Systems

Bogdan Aman and Gabriel Ciobanu

Romanian Academy, Institute of Computer Science, Iaşi, Romania
Blvd. Carol I no.8, 700505 Iaşi, Romania
bogdan.aman@gmail.com, gabriel@info.uaic.ro

Abstract. In this paper we consider the reversibility in the reaction systems, a bio-inspired formalism in which the reactions take place only if some inhibitors are not present. Forward reactions are exactly those of the previously defined reaction systems, while reverse reactions happen when a special symbol indicates a change in the environment. We study the controlled reversibility in reaction systems. The approach is illustrated by examples. Then the reversible reaction systems are translated into rewriting systems which are executable on the Maude software platform. Given such an implementation, several properties of the reversible reaction systems are verified.

1 Introduction

In this paper we investigate the reversibility of biochemical reactions in the framework of natural computing. Natural computing [16]) is the field of research dealing with models and computational techniques inspired by nature, helping us to understand the biochemical world around us in terms of information processing. Two important theories of natural computing inspired by the functioning of living cells are membrane computing [22] and reaction systems [13].

Membrane computing deals with multisets of symbols processed in the compartments of a membrane structure according to some multiset rewriting rules. The symbols are present with their multiplicity within the regions delimited by membranes. Some of these symbols evolve in parallel according to the rules associated with their membranes, while the others remain unchanged and can be used in the subsequent steps. The situation is different in reaction systems. These systems represent a qualitative model, and they deal with sets rather than multisets. Two major assumptions set the reaction systems apart from the membrane systems: (i) *threshold assumption* claiming that if a resource is present, then it is present in a "sufficient amount" such that several reactions needing the same reactant will not be in conflict (this means that reaction systems work with multisets having infinite multiplicity of their elements); (ii) *no permanency assumption* claiming that an entity will vanish from the current state unless it is produced by one of the reactions enabled in that state.

Reversible computation is an emerging paradigm extending the standard forwards-only mode of computation with the ability to execute also reversely, so that a computation can run backwards as naturally as it can go forwards. It has been studied for Turing machines [9, 21] and register machines [20]. A basic description of reversible computation is based on the possibility of returning to an initial state from any reachable state.

It is worth pointing out that there exist both uncontrolled and controlled reversibility. Uncontrolled reversibility means that reversing a system is done without indicating when a backward evolution is required. In chemical and biological systems, reversibility is natural and is related to the fault tolerance and stability of these systems. In these systems some operations are reversible only when there exist a specific context, an environmental modification which activate the computation in a desired direction (backward and forward). Looking to the biological systems where backward and forward evolutions depend on specific physical conditions, we consider a controlled reversibility in reaction systems. To control the reversibility, a special rollback symbol ρ is used to activate the backward evolution. We use specific sequences including this special symbol coming from the environment; these sequences control the direction of the computation in order to recover from failures or to avoid deadlocks, for instance.

Rewriting theory and rewriting logic [19] has been used for more than two decades as a computational framework able to express several paradigms. Computationally, rewriting logic represents a semantic framework in which many different models are naturally formalized; for instance, membrane systems can be executed and analyzed as rewrite theories [2, 6]. Logically, rewriting logic is a framework within which different logics can be represented. A more comprehensive list of calculi, programming languages, tools and applications implemented in Maude is presented in [19].

We present an implementation of the reversible reaction systems in the rewriting engine Maude based on a correspondence between their operational semantics and their rewriting logic translation. This allows the automated verification of their properties.

2 Reaction Systems

Reaction systems (abbreviated as RS) is a formal rewriting-like model of set evolution [10]. They are used for modelling processes driven by biochemical reactions; the fundamental idea in this framework is that the biochemical reactions are based on the mechanisms of facilitation and inhibition. Thus a reaction is modelled as a triplet: a set of reactants, a set of inhibitors, and a set of products. A reaction can take place in a given state if all its reactants are present in that state, and none of its inhibitors are present; when triggered, the reaction creates its products. We recall in what follows some elementary notions and notations about reaction systems, as they are available in the already published papers.

Let S be an alphabet (its elements are called molecules, or simply symbols). A reaction (over S) is a triple $a = (R, I, P)$, where R, I, P are nonempty subsets of S such that $R \cap I = \emptyset$. 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.

Given a configuration $T \subseteq S$ and a reaction $a \in rac(S)$, a is enabled by T (denoted by $a\ en\ T$) if $R_a \subseteq T$ and $I_a \cap T = \emptyset$. The result $res(a,T)$ of a on T is defined by $res(a,T) = P_a$. This reaction can be written as a rewrite of the form $T \xrightarrow{a} res(a,T)$. If a is not enabled by T, then $res(a,T) = \emptyset$; the fact that T cannot be rewritten by applying a is written as $T \overset{a}{\not\rightarrow}$.

If A is a finite set of reactions, then the result of A on T is defined by $res(A,T) = \bigcup_{a \in A} res(a,T)$. This can be written as $T \xrightarrow{A} res(A,T)$. The activity $en(A,T)$ of a set of reactions A on a finite set T is defined by $en(A,T) = \{a \in A : a\ en\ T\}$. Thus, $en(A,T)$ is the set of all reactions from A that are enabled by T. Note that $res(A,T) = res(en(A,T),T)$; this means that only the reactions from A which are enabled on T contribute to the result of A on T.

Definition 1. *A reaction system is an ordered pair $\mathcal{A} = (S,A)$, where S is an alphabet and $A \subseteq rac(S)$.*

The dynamic behaviour of the reaction systems is captured through the notion of an interactive process defined as follows.

Definition 2. *Let $\mathcal{A} = (S,A)$ be a reaction system. An interactive process in A is a pair $\pi = (\gamma, \delta)$ of finite sequences such that $\gamma = C_0, C_1, \ldots, C_{n-1}$, $\delta = D_1, \ldots, D_n$ 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, i.e., the state in which the interactive process is initiated, and the contexts $C_1, \ldots, C_n$ 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. This sequence formalizes the fact that we work with an open system interacting with the environment, somehow similar to what happens in the spiking neural P systems with input neurons [14]. 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

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

In the definition of the result of a set A of reactions on a set T of molecules, it is easy to note the two assumptions mentioned in the previous section: a molecule can evolve by means of several reactions (or can inhibit several reactions if it appears in inhibitor sets), hence the multiplicity of each molecule is unbounded, while all the molecules present at a given time "disappear" after the reactions are enabled and the computation continues with the set of molecules produced by the reactions.

Example 1. We describe the self-assembly of intermediate filaments from vimentin tetramers presented in [8] by using the reaction systems. Two tetramers (denoted by T) join to form an octamer (denoted by O), two octamers join to form a hexadecamer (denoted by H), while two hexadecamer join to form a unit length filament ULF. We consider the ULFs as elementary filaments (generically denoted by F). Two longer filaments join by end-to-end interactions, and form a longer complex. We present the molecular model of this basic representation and the corresponding reactions in the reaction system $\mathcal{A} = (S, A)$ with $S = \{T, O, H, F, d_I\}$:

Reaction in the molecular model	Reaction in the reaction system
$2T \to O$	$(\{T\}, \{d_I\}, \{O\})$
$2O \to H$	$(\{O\}, \{d_I\}, \{H\})$
$2H \to F$	$(\{H\}, \{d_I\}, \{F\})$
$2F \to F$	$(\{F\}, \{d_I\}, \{F\})$

The dummy variable d_I is used only to respect the constraint that the set of inhibitors should be non-empty.

A possible evolution of this system is into a loop after the second state, a loop from which every state contains all the species of the system:

State	C_i	D_i	W_i
0	$\{T\}$	$\emptyset$	$\{T\}$
1	$\{T\}$	$\{O\}$	$\{T, O\}$
2	$\{T\}$	$\{O, H\}$	$\{T, O, H\}$
3	$\{T\}$	$\{O, H, F\}$	$\{T, O, H, F\}$
4	$\{T\}$	$\{O, H, F\}$	$\{T, O, H, F\}$

This evolution is obtained if the context sequence γ has the form $\gamma = T^n$, namely $C_i = \{T\}$ for all $0 \leq i \leq n$.

Another possible evolution of the system is into a loop after the first state from which every state contains only the initial input:

State	C_i	D_i	W_i
0	$\{F\}$	$\emptyset$	$\{F\}$
1	$\emptyset$	$\{F\}$	$\{F\}$
2	$\emptyset$	$\{F\}$	$\{F\}$

This alternative evolution is obtained if the context sequence γ has the form $\gamma = T$, namely $C_0 = F$ and $C_i = \emptyset$ for all $1 \leq i \leq n$. As C_0 represents the initial state, it always hold that $C_0 \neq \emptyset$.

3 Reversible Reaction Systems

In order to have backward computations to the states W_i, we add a register T_i to keep track of the symbols that will disappear after each step and were not involved in the reactions of the current step (to assure the *no permanency assumption*). This is due to the fact that these symbols need to be recreated if we intend to reverse the computation. Thus, we define a register state $W'_i = (W_i, T_i)$, where $T_i \subseteq S \times \mathbb{N}$ is the set of objects disappearing during the evolution together with a number indicating how many steps ago they disappeared.

We can see a state W_i as an equivalence relation of register states obtained by ignoring the register. Namely, we can define a relation $\equiv$ on states given by $(W_i, T) = (W_i, T')$ for all $T, T' \subseteq S \times \mathbb{N}$. Clearly, $\equiv$ is an equivalence relation.

Proposition 1. *The set $S \times (S \times \mathbb{N})/_{\equiv}$ of equivalence classes is isomorphic with the set S of states of the reaction system $\mathcal{A}$.*

Proof. We construct $\phi : S \times (S \times \mathbb{N}) \to S$ inductively defined by $\phi((W, T)) = W$. This map induces a bijection $\overline{\phi} : S \times (S \times \mathbb{N})/_{\equiv} \to S$.

The evolution $W_0 \xrightarrow{E_0} W_1$ described in Section 2 becomes $(W_0, T_0) \xrightarrow{E_0} (W_1, T_1)$, where $T_0 = \emptyset$ and $T_1 = \bigcup_{t \in W_0 \setminus lhs(E_0)} (t, 0)$. In a similar manner, $W_i \xrightarrow{E_i} W_{i+1}$ for $i \geq 1$ becomes $(W_i, T_i) \xrightarrow{E_i} (W_{i+1}, T_{i+1})$, where $T_{i+1} = inc(T_i) \cup \bigcup_{t \in W_i \setminus lhs(E_i)} (t, 0)$ and $inc(T) = \bigcup_{(t,i) \in T} (t, i+1)$. The set $lhs(E) = \bigcup_{R,I,P \in E} R$ is the collection of all reactants from the set of enabled reactions E, while $W \setminus E$ is used to compute the set of molecules that vanishes after the reactions from E are applied to W.

To reverse a computation, a natural possibility is to reverse the reaction.

Reversing a reaction $a = (R_a, I_a, P_a)$ means that its reverse $\tilde{a}$ is able to undo the effects of a. According to [2], if the rule does not contain inhibitors, then $(P, \emptyset, R)$ is the reverse of $(R, \emptyset, P)$; this means to switch the position of reactants and products. After a step in which reaction a is applied, also its inhibitors I_a are modified by rules of A to $res(A, I_a)$, and the ones from $I_a \setminus res(A, I_a)$ vanish. To reverse this effect, the set of inhibitors of $\tilde{a}$ should be $(res(A, I_a) \cup (I_a \setminus res(A, I_a))$.

Definition 3. *The reverse of a reaction $a = (R_a, I_a, P_a)$ is given by the reaction $\tilde{a} = (P_a, (res(A, I_a) \cup (I_a \setminus res(A, I_a)), R_a)$. Similarly, the reverse of a set A of reactions is the set $\tilde{A} = \{\tilde{a} \mid a \in A\}$.*

As in Section 2 where for a reaction $a = (R, I, P)$ we imposed that $R \cap I = \emptyset$, the definition above has the problem that the reverse $\tilde{a} = (R', I', P')$ might not be a reaction because it might happen that $R' \cap I' \neq \emptyset$. For example, let us consider the reaction $a = (b, c, c)$, and assume that $res(A, c) = \emptyset$. Then the resulting reverse reaction would be $\tilde{a} = (c, c, b)$, which is not a reaction because reactants and inhibitors have a non-empty intersection. To overcome this problem, we impose from now on that we work only with reactions $a = (R, I, P)$ satisfying that $R \cap I = P \cap I = \emptyset$.

To avoid going backward and forward between two states for an infinite number of times, we also impose that the reverse computation is realized only when a special object ρ is introduced from the environment (context). The control is therefore performed by an (active) environment that provides at certain steps the special rollback symbol ρ signalling the system that it has to reverse its computation. This ρ is an abstraction of a physical reality in which a system is informed that a certain change in the environment has an effect on its evolution, as happens in heat shock response modelled previously by using the reaction systems in [7]. We assume also that this special symbol cannot be created by any reaction of A.

In this general framework, the evolution can take place by applying one of the following two rules:

$$\textit{(fwd)} \quad \frac{\rho \notin W_i}{(W_i, T_i) \xrightarrow{E_i} (W_{i+1}, T_{i+1})}$$

where $T_{i+1} = inc(T_i) \cup \bigcup_{t \in W_i \setminus lhs(E_i)}(t, 0)$, $inc(T) = \bigcup_{(t,i)\in T}(t, i+1)$ and $W_{i+1} = res(E_i, W_i)$;

$$\textit{(rev)} \quad \frac{\rho \in W_i}{(W_{i+1}, T_{i+1}) \overset{\tilde{E_i}}{\rightsquigarrow} (W_i, T_i)}$$

where $T_i = dec(T_{i+1})$, $dec(T_d) = \bigcup_{(t,i)\in T_d; i>0}(t, i-1)$, and $W_i = res(\tilde{E_i}, W_{i+1}) \cup zero(T_{i+1})$ with $zero(T) = \bigcup_{(t,0)\in T} t$. Also $\tilde{E_i} = en(\tilde{A}, W_{i+1})$.

This means that if $C_i \cap \{\rho\} = \emptyset$, then a forward computation takes place, while if $C_i \cap \{\rho\} \neq \emptyset$, then a backward computation takes place.

Example 2. To reverse a computation, we first construct the reversed reactions:

a	$\tilde{a}$
$(\{T\}, \{d_I\}, \{O\})$	$(\{O\}, \{d_I\}, \{T\})$
$(\{O\}, \{d_I\}, \{H\})$	$(\{H\}, \{d_I\}, \{O\})$
$(\{H\}, \{d_I\}, \{F\})$	$(\{F\}, \{d_I\}, \{H\})$
$(\{F\}, \{d_I\}, \{F\})$	$(\{F\}, \{d_I\}, \{F\})$

As d_I is a dummy variable, then is kept also in the reversed reactions.

We present now a possible evolution of the reaction system describing the self-assembly of intermediate filaments from vimentin tetramers by using also the reversed reactions when the special symbol ρ is appearing (but ignoring the registers that are empty for our example):

State	C_i	D_i	W_i
0	$\{T\}$	$\emptyset$	$\{T\}$
1	$\{T\}$	$\{O\}$	$\{T, O\}$
2	$\{\rho\}$	$\{O, H\}$	$\{\rho, O, H\}$
3	$\{\emptyset\}$	$\{T, O\}$	$\{T, O\}$
4	$\{T\}$	$\{O, H\}$	$\{T, O, H\}$
5	$\{T\}$	$\{O, H, F\}$	$\{T, O, H, F\}$
6	$\{\rho\}$	$\{O, H, F\}$	$\{\rho, O, H, F\}$
7	$\emptyset$	$\{T, O, H, F\}$	$\{T, O, H, F\}$

It can be easily noticed that if the rollback symbol ρ is introduced from the environment, then the system reaches the previous state going backward.

Another possible evolution of the system is:

State	C_i	D_i	W_i
0	$\{F\}$	$\emptyset$	$\{F\}$
1	ρ	$\{F\}$	$\{\rho, F\}$
2	ρ	$\{H, F\}$	$\{\rho, H, F\}$
3	ρ	$\{O, H, F\}$	$\{\rho, O, H, F\}$
4	$\emptyset$	$\{T, O, H, F\}$	$\{T, O, H, F\}$
5	$\emptyset$	$\{O, H, F\}$	$\{O, H, F\}$
6	$\emptyset$	$\{H, F\}$	$\{H, F\}$
7	$\emptyset$	$\{F\}$	$\{F\}$

In this case it is worth noting that even if we start from a system containing only F, the forward evolution keeps the same state while the backward evolution reaches the state $\{T, O, H, F\}$ from which going forward we reach again the state F (whenever the environment does not offer other rollback symbols).

Remark 1. For the purpose of this paper it is enough to consider the controlled reversibility by using context sequences with symbols from $\{\emptyset; \rho\}$. The case of considering scenarios using symbols from an extended set represents a further work.

Our reversible reaction systems (RRS) represent only a decoration of the reaction systems (RS) defined in the previous sections. In fact, as for the most of the existing reversible calculi, such decorations can be erased by a forgetful map $\phi : RRS \to RS$ defined as $\phi((W_i, T_i)) = W_i$. Conversely, one can lift any RS configuration to an RRS configuration by using the map $l : RS \to RRS$ defined by $l(W_i) = (W_i, \emptyset)$, namely by adding an empty register to a state.

It is enough to forget about backward rules by considering $C_i = \emptyset$ in the reversible reaction systems. In this way, there is no object ρ coming from the environment to inhibit the forward rules. This is formally stated in what follows; the next result shows that a step in the initial reaction system can be modelled by a forward step in the reversible reaction system.

Proposition 2. $W \xrightarrow{E} W'$ *if and only if* $(W, T) \xrightarrow{E} (W', T')$.

Proof. $\Rightarrow$: If $W \xrightarrow{E} W'$, then $W' = res(E, W)$. By the previous constructions, for each configuration W of a reaction system there exist a corresponding configuration of a reversible reaction system constructed by $l(W) = (W, \emptyset)$. By applying the rule *(fwd)* to (W, T) where $T = \emptyset$, we obtain the configuration (W'', T) where $W'' = res(E', W)$ and $T = \bigcup_{t \in W \setminus lhs(E)} (t, 0)$. Due to the threshold assumption of the reaction systems, we got that $E = E'$, namely there is an unique set of reactions applicable to W. This means that $W'' = W'$, and so $(W, T) \xrightarrow{E} (W', T')$ holds.

$\Leftarrow$: If $(W, T) \xrightarrow{E} (W', T')$, then $T' = \cup_{(t,i) \in T} (t, i+1) \cup \bigcup_{t \in W \setminus lhs(E)} (t, 0)$ and $W' = res(E, W)$. By applying the forgetful map ϕ to the configuration (W, T) of the reversible reaction systems, we obtain the configuration W of the reaction systems. By applying all the possible reactions to this configuration, we obtain

$W \xrightarrow{E'} W''$ where $W'' = res(E', W)$. Due to the threshold assumption of the reaction systems, we got that $E = E'$, namely there is a unique set of reactions applicable to W. This means that $W'' = W'$, and so $W \xrightarrow{E} W'$ holds. $\square$

Remark 2. Note that by using a reaction system, it is possible to reverse a computation beyond its initial state. This means that there are cases in which the construction provided before produces, by going backward from a state (W, T), a state (W', T') in which W' is not contained in the set of pre-images of W. Our approach is similar with the one presented in [17], where a process calculus for the out-of-causal order reversible computation was proposed. This approach is illustrated by the last case of Example 2, where starting from $\{F\}$ and going backward some steps without performing any forward step previously, we reach a new state $\{T, O, H, F\}$.

The following two theorems show that, in certain cases, if the current state is related to the obtained one by certain relations, then the reversible reaction systems enjoy a standard property of reversible calculi described by so-called *loop lemma* in [12]: backward reductions are the inverse of the forward ones, and vice-versa. It is worth noting that the following two theorems provide necessary conditions such that the reverse reactions defined as in Definition 3 are able to provide, together with the additional memory, a way of "going back" one step in the computation (i.e., a causal reversibility).

Theorem 1. *If $W = res(\tilde{E}, W') \cup zero(T')$ and $\rho \in W'$, then*
$$(W,T) \xrightarrow{E} (W',T') \text{ implies } (W',T') \overset{\tilde{E}}{\rightsquigarrow} (W,T).$$

Proof. If $(W,T) \xrightarrow{E} (W',T')$, then $T' = inc(T) \cup \bigcup_{t \in W \setminus lhs(E)} (t, 0)$, $inc(T) = \bigcup_{(t,i) \in T} (t, i+1)$ and $W' = res(E, W)$. Since $\rho \in W'$, then a *(rev)* rule can be applied, and so $(W',T') \overset{\tilde{E}}{\rightsquigarrow} (W'',T'')$ with $T'' = dec(T')$ and $dec(T') = \bigcup_{(t,i) \in T'; i>0} (t, i-1)$, as well as $W'' = res(\tilde{E}, W') \cup zero(T')$ with $zero(T') = \bigcup_{(t,0) \in T'} t$. Notice that $T'' = dec(T') = dec(inc(T) \cup \bigcup_{t \in W \setminus lhs(E)} (t,0)) = dec(inc(T)) \cup dec(\bigcup_{t \in W \setminus lhs(E)} (t,0)) = T \cup \emptyset = T$, and also due to the hypothesis that $W'' = res(\tilde{E}, W') \cup zero(T') = W$. This means that we got $(W',T') \overset{\tilde{E}}{\rightsquigarrow} (W,T)$, as desired. $\square$

Theorem 2. *If $W' = res(E, W)$ and $\rho \notin W$, then*
$$(W',T') \overset{\tilde{E}}{\rightsquigarrow} (W,T) \text{ implies } (W,T) \xrightarrow{E} (W',T').$$

Proof. If $(W',T') \overset{\tilde{E}}{\rightsquigarrow} (W,T)$, then $T = dec(T')$ where $dec(T') = \bigcup_{(t,i) \in T'; i>0} (t, i-1)$ and $W = res(\tilde{E}, W') \cup zero(T')$ with $zero(T') = \bigcup_{(t,0) \in T'} t$. Since $\rho \notin W$, then a *(fwd)* rule can be applied, and so $(W,T) \xrightarrow{E} (W'',T'')$ with $T'' = inc(T) \cup \bigcup_{t \in W \setminus lhs(E)} (t, 0)$, $inc(T) = \bigcup_{(t,i) \in T} (t, i+1)$ and $W'' = res(E, W)$. It should be noticed that $T'' = inc(T) \cup \bigcup_{t \in W \setminus lhs(E)} (t, 0)$

$= inc(dec(T')) \cup \bigcup_{t \in (res(\tilde{E},W') \cup zero(T')) \setminus lhs(E)} (t,0)$
$= (T' \setminus \bigcup_{t \in zero(T')} (t,0)) \cup \bigcup_{t \in (rhs(\tilde{E}) \cup zero(T')) \setminus lhs(E)} (t,0)$
$= (T' \setminus \bigcup_{t \in zero(T')} (t,0)) \cup \bigcup_{t \in (lhs(E) \cup zero(T')) \setminus lhs(E)} (t,0)$
$= (T' \setminus \bigcup_{t \in zero(T')} (t,0)) \cup \bigcup_{t \in zero(T')} (t,0) = T'$
and also due to the hypothesis that $W'' = res(E,W) = W'$. This means that we got $(W,T) \xrightarrow{E} (W',T')$, as desired. □

4 Implementation of Reversible Reaction Systems

Rewriting logic is a computational logic which combines equational logic with term rewriting. According to [11], a *rewrite theory* is a triple (Σ, E, R) , where Σ is a signature of function symbols, E a set of (possibly conditional) Σ-equations, and R a set of (possibly conditional) Σ-rewrite rules. The conditions for a rewrite rule can involve both equations and rewrite rules. Generally, a typed setting is used in [18] under the form of an order-sorted equational logic (Σ, E) which has sorts, subsort inclusions and kinds (connected components of sorts). The notation $\mathcal{R} \vdash t \to t'$ is used to express that $t \to t'$ is provable in the theory $\mathcal{R}$ using the inference rules of rewriting logic. For a kind k and a set of kinded variables X, $T_\Sigma(X)_k$ denotes the set of Σ-terms of kind k over the variables in X. If s is a sort in the kind k, $T_\Sigma(X)_s$ is the subset of $T_\Sigma(X)_k$ consisting of Σ-terms of sort s over X. Given a rewrite theory $\mathcal{R} = (\Sigma, E, R)$, the sentences which $\mathcal{R}$ proves are of form $(\forall X) t \to t'$, with $t, t' \in T_\Sigma(X)_k$ for some kind k. These sentences are obtained from the following inference rules:

reflexivity For each $t \in T_\Sigma(X)$, $\dfrac{}{\mathcal{R} \vdash t \to t}$

equality $\dfrac{(\forall X) u \to v, E \vdash u = u', E \vdash v = v'}{(\forall X) u' \to v'}$

congruence For each $f \in \Sigma_{s_1 \ldots s_n, s}$, $t_i \in T_\Sigma(X)_{s_i}$

$$\frac{(\forall X) t_j \to t'_j, j \in J \subseteq [n]}{(\forall X) f(t_1, \ldots, t_n) \to f(t'_1, \ldots, t'_n)}, \text{ where } t'_i := t_i \text{ whenever } i \notin J;$$

replacement For each $\theta : X \to T_\Sigma(Y)$ and for each rule in $\mathcal{R}$ of the form

$(\forall X) t \to t'$ if $(\bigwedge_i u_i = u'_i) \wedge (\bigwedge w_j \to w'_j)$, we have

$$\frac{\bigwedge_x (\forall Y) \theta(x) \to \theta'(x)) \wedge (\bigwedge_i (\forall Y) \theta(u_i) = \theta(u'_i)) \ \wedge (\bigwedge_j (\forall Y) \theta(w_j) \to \theta(w'_j))}{(\forall Y) \theta(x) \to \theta(x')}$$

where θ' is the substitution obtained from θ by some rewritings $\theta(x) \to \theta'(x)$ for each $x \in X$;

transitivity $\dfrac{(\forall X) t_1 \to t_2, (\forall X) t_2 \to t_3}{(\forall X) t_1 \to t_3}$.

The notation $\mathcal{R} \vdash t \to t'$ is used to express that $t \to t'$ is provable in the theory $\mathcal{R}$ by using the above inference rules.

In what follows we use the rewriting engine Maude to describe a rewriting theory corresponding to the semantics of the reversible reaction systems. In order to translate the syntax of reversible reaction systems, we use several sorts with easy-to-understand names: e.g., `ESymbol` is used to represent symbols from the environment. Between the given sorts there exist some subsorting relations, from which we mention `subsorts ESymbol < ESymbols` illustrating the fact that an environment symbol is part of a set of environment symbols. The sort `TSymbol` is used to represent timed symbols, namely counting the number of steps since a symbol was removed from the system.

```
sorts Symbol Symbols ESymbol ESymbols TSymbol TSymbols
      RState Reaction Reactions .
subsorts Symbol < ESymbol Symbols .
subsorts Symbols < ESymbols .
subsorts ESymbol < ESymbols .
subsorts TSymbol < TSymbols .
subsorts Reaction < Reactions .
```

To represent the symbols, reactions and states of the reaction systems, we use the constructors described below. The sets of symbols and reactions are described by using associative and commutative constructors.

```
op es : -> Symbol [ctor].
op _ | _ : Symbols Symbols -> Symbols [assoc comm id: es] .
op rho : -> ESymbol [ctor] .
op fw : -> ESymbol [ctor] .
op _ ~ _ : ESymbols ESymbols -> ESymbols [assoc] .
op et : -> TSymbol [ctor].
op _ || _ : TSymbols TSymbols -> TSymbols [assoc comm id: et] .
op [_ , _] : Symbols Nat -> TSymbols .
op {_ , _ , _} : Symbols Symbols Symbols -> Reaction .
op er : -> Reaction [ctor].
op _ ; _ : Reactions Reactions -> Reactions [assoc comm id: er] .
op < _ $ _ $ _ $ _ > : ESymbols Symbols TSymbols Reactions -> RState .
```

The reactions are simulated as conditional rewrite rules:

```
crl [Fwd] : < fw ~ E $ X $ ts $ A > =>
            < E $ res(A , X) $ inc(ts) || addtime(rem(lhs(A), X)) $ A >
                 if en(A, X) =/= er .

crl [FwdFin] : < fw $ X $ ts $ A > =>
            < es $ res(A , X) $ inc(ts) || addtime(rem(lhs(A), X)) $ A >
                 if en(A, X) =/= er .

crl [Rev] : < rho ~ E $ X $ ts $ A > =>
            < E $ res(rev(A , A) , X) | zero(ts) $ dec(ts) $ A >
                 if en(rev(A , A), X) =/= er .

crl [RevFin] : < rho $ X $ ts $ A > =>
            < es $ res(rev(A , A) , X) | zero(ts) $ dec(ts) $ A >
                 if en(rev(A , A), X) =/= er .

crl [FwdStop] : < fw ~ E $ X $ ts $ A > =>
            < E $ es $ inc(ts) || addtime(X) $ A >
                 if en(A, X) == er /\ X =/= es .

crl [RevStop] : < rho ~ E $ X $ ts $ A > =>
            < E $ es $ inc(ts) || addtime(X) $ A >
                 if en(A, X) == er /\ X =/= es .
```

It should be noticed that there are three instances for each of the reactions *(fwd)* and *(rev)*. The need for these three rules is due to the fact that either the input provided by the environment ends or the system cannot evolve anymore. The functions `res`, `addtime`, `rem`, `lhs`, `en`, `zero`, `inc` and `dec` are used to compute the next configurations and to test if rules are applicable. They are defined similar with the ones used in the reactions *(fwd)* and *(rev)*.

The correspondence between the operational semantics of the reversible reaction systems on one hand and the rewrite theory on the other hand is given by a mapping ψ :RRS $\to$ RState defined inductively by

$$\psi((W_i, T_i), A) = < C_i \sim C_{i+1} \ldots \sim Cn\$D_i\$T_i\$A > ,$$

where $W_i = C_i \cup D_i$ and $C_{i+1} \ldots \sim Cn$ represents the remaining of the external input. For simplification and a more straightforward translation, we could use a function that after using the current C_i will provide the next input from the environment.

By $\mathcal{R}_{\mathcal{RRS}}$ we denote the rewrite theory defined by the rewrite rules `[Fwd]`, `[FwdFin]`, `[Rev]`, `[RevFin]`, `[FwdStop]` and `[RevStop]` together with the operators and equations defining them. The next theorem proves the correspondence between the dynamics of a reversible reaction system and its rewrite theory.

Theorem 3. $(W,T) \xrightarrow{E} (W',T')$ *iff* $\mathcal{R}_{RRS} \vdash \psi((W,T),A) \Rightarrow \psi((W',T'),A)$.

Example 3. We provide a small example of a reversible reaction system, and then analyze it by using the Maude implementation. We can verify that the rules are applied properly, and the results are the desired ones.

Consider the following reversible reaction system.

```
< fw ~ fw ~ rho ~ fw  $ (c|d|f) $ et $ ({c,d1,d} ; {f,d1,c}) > .
```

When using the rewrite command `rew` on the above system, Maude executes the specification by applying the previously presented rules and equations, and finally returns the output below. Since sometime we are not interested to display all the steps and states, the command `rew [n]` can be used to obtain systems reachable in n steps:

```
rewrite [1] in RS-EXAMPLE : < fw ~ fw ~ rho ~ fw $ c | d | f $ et $
        {c,d1,d} ; {f,d1,c} > .
rewrites: 121 in 0ms cpu (0ms real) (~ rewrites/second)
result RState: < fw ~ rho ~ fw $ c | d $ [d,0] $
        {c,d1,d} ; {f,d1,c} >
==========================================
rewrite [2] in RS-EXAMPLE : < fw ~ fw ~ rho ~ fw $ c | d | f $ et $
        {c,d1,d} ; {f,d1,c} > .
rewrites: 304 in 0ms cpu (0ms real) (~ rewrites/second)
result RState: < rho ~ fw $ d $ [d,0] || [d,1] $
        {c,d1,d} ; {f,d1,c} >
==========================================
rewrite [3] in RS-EXAMPLE : < fw ~ fw ~ rho ~ fw $ c | d | f $ et $
        {c,d1,d} ; {f,d1,c} > .
rewrites: 569 in 4ms cpu (1ms real) (142250 rewrites/second)
result RState: < fw $ c | d $ [d,0] $  {c,d1,d} ; {f,d1,c} >
==========================================
rewrite [4] in RS-EXAMPLE : < fw ~ fw ~ rho ~ fw $ c | d | f $ et $
              {c,d1,d} ; {f,d1,c} > .
rewrites: 696 in 0ms cpu (1ms real) (~ rewrites/   second)
result RState: < es $ d $ [d,0] || [d,1] $ {c,d1,d} ; {f,d1,c} >
```

It is easy to notice that, ignoring the context symbols, the configurations after one and three rewrites and after two and four rewrites are equal, meaning that the reversing evolution works as desired.

5 Conclusion

Membrane computing and reaction systems are branches of natural computing aiming to define computing models from the structure and functioning of the living cell. Membrane systems represent a quantitative model of (controlled) multiset rewriting [22]. On the other hand, reaction systems [13] represent a qualitative model of set rewriting. Some research comparing them was done in [23], while in [4] membrane systems with no-persistence assumption of reaction systems from the viewpoint of the computational power were considered.

Reversible membrane systems were considered in [15], but the model does not uses maximal parallel rewriting; the main result is the simulation of the Fredkin gate, and so it actually studies the reversible circuits. In [5] it is studied the reversibility of membrane systems with maximal parallelism systems only from a computability point of view. The so-called *dual* P systems [1] present reversibility in membrane systems as duality (under the influence of category theory). In [3] it is presented a more detailed description of this kind of reversibility in membrane systems.

In this paper we present a controlled reversibility in the context of reaction systems. An important aspect of this approach is given by considering additional reversing reactions to the initial set of reactions with inhibitors, as well as by adding an external control by means of a special symbol ρ informing the system that a rollback is needed. Specific results (including so-called *loop* results) are proved, as well as an operational correspondence between reaction systems and rewriting theory. This operational correspondence allows to translate the reversible reaction systems into rewriting systems which are executable in Maude. Given such an implementation, several properties of the reversible reaction systems can be verified.

References

1. O. Agrigoroaiei, G. Ciobanu. Dual P Systems. *Lecture Notes in Computer Science* **5391**, 95–107 (2009).
2. O. Agrigoroaiei, G. Ciobanu. Rewriting Logic Specification of Membrane Systems with Promoters and Inhibitors. *Electronic Notes in Theoretical Computer Science* **238**, 5–22 (2009).
3. O. Agrigoroaiei, G. Ciobanu. Reversing Computation in Membrane Systems. *Journal of Logic and Algebraic Programming* **79**, 278–288 (2010).

4. A. Alhazov, B. Aman, R. Freund, S. Ivanov. Simulating R Systems by P Systems. *Lecture Notes in Computer Science* **10105**, 51–66 (2017).
5. A. Alhazov, K. Morita. On Reversibility and Determinism in P System. *Lecture Notes in Computer Science* **957**, 158–168 (2010).
6. O. Andrei, G. Ciobanu, D. Lucanu. Executable Specifications of P Systems. *Lecture Notes in Computer Science* **3365**, 126–145 (2005).
7. S. Azimi, B. Iancu, I. Petre. Reaction System Models for the Heat Shock Response. *Fundamenta Informaticae* **131**(3-4), 299–312 (2014).
8. S. Azimi, C. Panchal, E. Czeizler, I. Petre. Reaction Systems Models For the Self-assembly of Intermediate Filaments. *Annals of University of Bucharest*, **LXII**(2), 9–24 (2015).
9. C.H. Bennett. Logical Reversibility of Computation. *IBM Journal of Research and Development* **17**, 525-532 (1973).
10. R. Brijder, A. Ehrenfeucht, M.G. Main, G. Rozenberg. A Tour of Reaction Systems. *Int'l Journal of Foundations of Computer Science* **22**(7), 1499–1517 (2011).
11. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C.L. Talcott. *All About Maude - A High Performance Logical Framework. How to Specify, Program and Verify Systems in Rewriting Logic.* Springer (2007).
12. V. Danos, J. Krivine. Reversible Communicating Systems. *Lecture Notes in Computer Science* **3170**, 292–307 (2004).
13. A. Ehrenfeucht, G. Rozenberg. Reaction Systems. *Fundamenta Informaticae* **75**(1), 263–280 (2007).
14. M. Ionescu, Gh. Păun, T. Yokomori. Spiking Neural P Systems. *Fundamenta Informaticae* **71** (2), 279–308 (2006).
15. A. Leporati, C. Zandron, G. Mauri. Reversible P Systems to Simulate Fredkin Circuits. *Fundamenta Informaticae* **74**(4), 529-548 (2006).
16. L. Kari, G. Rozenberg. The Many Facets of Natural Computing. *Communications of the ACM* **51**, 72–83 (2008).
17. S. Kuhn, I. Ulidowski. A Calculus for Local Reversibility. *Lecture Notes in Computer Science* **9720**, 20–35 (2016).
18. J. Meseguer. Membership Algebra as a Logical Framework for Equational Specification. *Lecture Notes in Computer Science* **1376**, 18–61 (1997).
19. J. Meseguer. Twenty Years of Rewriting Logic. *Journal of Logic and Algebraic Programming* **81**(7-8), 721–781 (2012).
20. K. Morita. Universality of a Reversible Two-Counter Machine. *Theoretical Computer Science* **168**, 303-320 (1996).
21. K. Morita, Y. Yamaguchi. A Universal Reversible Turing Machine, *Lecture Notes in Computer Science* **4664**, 90–98 (2007).
22. Gh. Păun. Computing With Membranes. *Journal of Computer and System Sciences* **61**, 108–143 (1998).
23. Gh. Păun and M.J. Pérez-Jiménez. Towards Bridging Two Cell-Inspired Models: P Systems and R Systems. *Theoretical Computer Science* **429**, 258–264 (2012).

Multiset Patterns and their Application to Dynamic Causalities in Membrane Systems

Roberto Barbuti, Roberta Gori and Paolo Milazzo

Dipartimento di Informatica, Università di Pisa
Largo Pontecorvo 3, 56127 Pisa, Italy.
Email: {*barbuti,gori,milazzo*}*@di.unipi.it*

Abstract. In this paper we investigate dynamic causalities in membrane systems by proposing the concept of "predictor", originally defined in the context of Ehrenfeucht and Rozemberg's reaction systems. The goal is to characterize sufficient conditions for the presence of a molecule of interest in the configuration of a P system after a given number of evolution steps (independently from the non-deterministic choices taken). Such conditions can be used to study causal relationships between molecules. To achieve our goal, we introduce the new concept of "multiset pattern" representing a logical formula on multisets. A predictor can be expressed as a pattern characterizing the initial multisets that will surely lead (sufficient condition) to the presence of the molecule of interest after the given number of evolution steps. We define also an operator that computes such a predictor. The pattern obtained from the operator is a sound predictor, but, in general, not a complete predictor, that is, it is not able to characterize *all* the initial multisets that surely evolve after k steps in multisets containing the molecule of interest.

1 Introduction

The understanding of causal relationships among the events happening in a biological (or bio-inspired) system is an issue investigated in the context both of systems biology (see e.g. [14,7,8]) and of natural computing (see e.g. [12]).

In [11] Brijder, Ehrenfeucht and Rozenberg initiate an investigation of *causalities* in reaction systems [13,10]. Causalities deal with the ways entities of a reaction system influence each other. In [11], both static/structural causalities and dynamic causalities are discussed, introducing the idea of *predictor*. A predictor can be used to determine whether a molecule of interest s will be produced after k steps of execution of the reaction system, without executing the system itself.

The environment is the only source of non-determinism in a reaction system. Knowledge about the molecules which will be provided at each step by the environment is necessary to determine whether a molecule s will be produced after k steps. Moreover, not all molecules are relevant for the production of a molecule of interest s. On the basis of these two observations, a predictor is defined as the subset of molecues Q whose supply by the environment should be observed in order to determine whether s will be produced or not after k steps.

In [3,4,5] we pushed forward the idea of predictors by defining the notion of *formula based predictor*. A formula based predictor consists in a propositional logic formula to be satisfied by the sequence of sets of molecules provided by the environment to the reaction system. Such a logic formula precisely discriminates the cases in which a particular molecule s will be produced after a given number of steps from the cases in which it will not.

P systems [15] are much more powerful and complex than reaction systems. They are based on multisets rather than sets and evolution rules are applied with *maximal parallelism* and with a non-deterministic competition for reactants.

The behaviour of a P system is determined only by the initial multiset and by the non-deterministic choices made at each maximally parallel step. In this context, a notion of predictor may correspond to a logical formula to be satisfied by the initial multiset (representing either a *sufficient condition* or a *necessary condition*) for a molecule of interest s to be present after a given number of evolution steps. Sufficient and necessary conditions have to be dealt with separately due to the intrinsically non-determinsitic nature of P systems.

In this paper we propose notion of *multiset pattern* as a way to express a logical formula on multisets: if the initial multiset of the P systems satisfies (*matches*) a given pattern, the molecule of interest will be *for sure* present after k steps; nothing can be said otherwise (sufficient condition). Moreover, we will define an operator that recursively computes the pattern for the production of a molecule of interest (actually, a multiset of molecules of interest) in k steps. The pattern obtained from the operator will be a sound predictor, but, in general, not a complete predictor. This means that there can be multisets that do not match the pattern, but that always lead to the presence of the molecules of interest in k steps.

For lake of space here we focus on sufficient conditions since necessary conditions are more standard and could be derived from the concepts we introduce.

2 Preliminaries

2.1 Multisets

We denote by $\mathbb{N}$ the set of natural numbers. Let U be an arbitrary set. A *multiset* (over U) is a mapping $M : U \to \mathbb{N}$; with $|M|_a$, for $a \in U$, we denote the *multiplicity* of a in the multiset M. The *support* of a multiset M is the set $supp(M) = \{a \mid |M|_a > 0\}$. A multiset is empty when its support is empty and it is denoted by $\emptyset$.

In order to distinguish multiset operations from standard set operations we use the following notations: $\subseteq^*$ for multiset inclusion, $\cup^*$ for multiset union, $\cap^*$ for multiset intersection. For multisets defined over a set of molecules (an alphabet) V, we will use also the standard string representation, with ϵ representing the empty multiset and V^* representing the set of all multisets over V. With $\wp(V^*)$ we indicate the power set of the set V^*.

2.2 Membrane Systems

In this paper we consider *flat* P systems, namely in which the membrane structure consists only of the skin membrane. Flat P systems are defined as follows.

Definition 1. *A* flat P system *is a construct* $\Pi = (V, w, R)$ *where:*

– V *is an* alphabet *whose elements are called* molecules*;*
– $w \in V^*$ *is the* initial multiset*;*
– R *is a finite set of* evolution rules.

From [6] it follows that a P system with a standard membrane structure can be translated into an equivalent P system having a (flat) membrane structure that consists only of the skin.

In this paper we assume P systems to be closed computational devices, namely we assume that molecules cannot be sent out of the skin membrane (i.e. rules sending molecules out are not allowed in the skin membrane) and molecules cannot be received by the skin membrane from outside.

As a consequence, evolution rules will have the simple form $u \rightarrow v$ with $u, v \in V^*$. We denote with $\mathcal{R}$ the set of all evolution rules.

Given an evolution rule $r = u \rightarrow v$, we denote with $react(r)$ and $prod(r)$ its multisets of reactants u and products v, respectively. The same notations extend naturally to multisets of rules: given a multiset of rules $M \in \mathcal{R}^*$, we have $react(M) = \bigcup^*_{r \in supp(M)} react(r)^{|M|_r}$ and $prod(M) = \bigcup^*_{r \in supp(M)} prod(r)^{|M|_r}$.

We assume evolution rules to be applied with standard maximal parallelism. Since we also assumed P systems not to send/receive molecules to/from the external environment, we obtain that the behaviour of a P system is determined only its initial multiset and by the non-deterministic choices made at each maximally parallel step.

3 Multiset Patterns

3.1 Definition

Given an alphabel V, a multiset pattern expresses a condition on multisets in V^*. A *basic* multiset pattern is denoted by a pair $(u, \{u_1, ..., u_n\})$ where $u, u_1, ..., u_n$ are multisets in V^*. More complex patterns can be obtained by composing basic patterns by using propositional logic connectives $\wedge$ and $\vee$. The syntax of multiset patterns is defined as follows.

Definition 2 (Multiset Patterns). *Given an alphabet V, $\mathcal{P}_V$ is the set of multiset patterns on V^* inductively defined as follows:*

– $\{true, false\} \in \mathcal{P}_V$*,*
– *if* $p \in (V^* \times \wp(V^*))$ *then* $p \in \mathcal{P}_V$*,*
– *if* $p_1, p_2 \in \mathcal{P}_V$ *then* $p_1 \vee p_2, p_1 \wedge p_2 \in \mathcal{P}_V$*.*

When the reference alphabet V is clear from the context, we will denote the set $\mathcal{P}_V$ simply as $\mathcal{P}$.

Now, we formally define the notion of satisfaction of a pattern by a multiset, that is the semantics of multiset patterns. The idea is that a multiset w satisfies a basic pattern $(u, \{u_1, ..., u_n\})$ if by removing from w the multisets $u_1, ..., u_n$ in any maximal way (i.e. so that what remains does not include any of $u_1, ..., u_n$) we always obtain a multiset that includes u. For example, multiset A^4B^2 satisfies the pattern $(A, \{AB\})$ since after removing AB a maximal number of times (two times) we obtain A^2 that includes A.

The semantics of multiset patterns is formally defined as follows.

Definition 3. *Given an alphabet V, the* satisfaction relation $\models\ \subseteq V^* \times P_V$ *is the smallest relation inductively defined as follows:*

$$\begin{array}{ll} w \models \mathit{true} & \\ w \models (u, \{u_1, ..., u_n\}) & \textit{iff } \forall o_1, ...o_n \in \mathbb{N} \textit{ such that } w \supseteq^* u_1^{o_1} u_2^{o_2} \dots u_n^{o_n}, \\ & \textit{it holds } w \supseteq^* u u_1^{o_1} u_2^{o_2} \dots u_n^{o_n} \\ w \models p_1 \wedge p_2 & \textit{iff } w \models p_1 \textit{ and } w \models p_2 \\ w \models p_1 \vee p_2 & \textit{iff either } w \models p_1 \textit{ or } w \models p_2 \end{array}$$

For the case of a basic pattern $(u, \{u_1, ..., u_n\})$, Definition 3 includes a requirement on all possible values $o_1, ..., o_n \in \mathbb{N}$ used as multiplicities of the occurrences of $u_1, ..., u_n$ in w. It is easy to see that the requirement can be checked by considering only the maximal combinations of values $o_1, ..., o_n$ such that $w \supseteq^* u_1^{o_1} u_2^{o_2} \dots u_n^{o_n}$.

3.2 Multiset Patterns and Multiset Languages

Maximal patterns can be used to express complex conditions on multisets. For example, the pattern $p_1 = (AB, \{BC, BE\})$ is satisfied by multisets that contain at least one A and one B, and where the sum of the numbers of C and of E is smaller than the number of B. Indeed, the set of multisets satisfying the pattern corresponds to the multiset language

$$L_{p_1} = \{w \in V^* \mid |w|_A > 0, |w|_B > 0, |w|_B > |w|_C + |w|_E\}.$$

Patterns can express also conditions that are not in the "greater than" form. For example, pattern $(A, \{AA\})$ is satisfied by multisets with an odd number of A. Namely, it characterizes the language

$$L_{p_2} = \{w \in V^* \mid |w|_A \equiv 1 (mod\ 2)\}.$$

More generally, given any $n \in \mathbb{N}$, the multiset pattern $(A, \{A^n\})$ is such that $w \models (A, \{A^n\})$ iff w satisfies $|w|_A \not\equiv 0 (mod\ n)$.

The examples we have given show that multiset patterns could be used to characterize multiset languages. It could be interesting to investigate the classes of languages characterized by multiset patterns. Although such an investigation is out of the scope of this paper, we give a few more examples of patterns characterizing interesting languages.

Let us look for a pattern characterizing the language consisting only of the A^3 multiset. The pattern $p_3 = (A^3, \{\})$ would not be good, since it is satisfied by any multiset with *at least* three instances of A. We could combine p_3 with a pattern satisfied by multiset containing *at most* three instances of A. The latter pattern can be obtained by considering an additional molecule O, assumed to be present only once in the multiset, as follows: $p_4 = (O, \{OA^4\})$. By combining the two patterns (and by generalizing them to any fixed value $k \in \mathbb{N}$) we obtain $p_5 = (A^k, \{\}) \wedge (O, \{OA^{k+1}\})$ that is actually equivalent to $p_5' = (OA^k, \{OA^{k+1}\})$. The characterized language is

$$L_{p_5} = \{w \in V^* \mid |w|_O > 0, |w|_A = k + (|w|_O - 1)(k+1)\}$$

that, by the assumption $|w|_O = 1$, becomes

$$L'_{p_5} = \{w \in V^* \mid |w|_O = 1, |w|_A = k\}.$$

A few more examples: $(A, \{AB\})$ characterizes the language $A^m B^n$ with $m > n$. Consequently, $(A, \{AB\}) \vee (B, \{AB\})$ characterizes the language $A^m B^n$ with $m \neq n$. It seems not possible to define a pattern that characterizes the complement of the previous language, namely $A^n B^n$, even if we consider additional molecules (like O in a previous example). This would be made possible for instance by including logical negation in the syntax of patterns. Also the extension of multiset patterns with priorities associated to multisets to be consumed in a maximal way could have enough expressive power to characterize $A^n B^n$.

3.3 Simplification of Multiset Patterns

Multiset patterns express logical conditions on multisets, hence they can be simplified using standard logic rules. For instance, a conjunction of basic patterns can be simplified to $false$ every time the basic patterns implicitly express opposite constraints. Moreover, the following properties of the satisfaction relation $\models$ defined in Definition 3 allow us to consider further simplification rules for multiset patterns.

Lemma 1. *Let $(w, \{u_1, ..., u_n\})$ be a basic multiset pattern and $v \in V^*$. Then $v \models (w, \{u_1, ..., u_n\})$ iff $v \models (w, \{u_i \mid u_i \cap^* w \neq \emptyset\})$.*

Using the previous result a basic pattern $(w, \{u_1, ..., u_n\})$ can be always simplified into $(w, \{u_i \mid u_i \cap^* w \neq \emptyset\})$.

Lemma 2. *Let $(w, \{u_1, ..., u_n\})$ be a basic pattern. If there exists $i \in \{1, ..., n\}$ such that $u_i \subseteq^* w$, then for all $v \in V^*$ it holds $v \not\models w\{u_1, ..., u_n\}$.*

Using the previous result a basic pattern $(w, \{u_1, ..., u_n\})$ such that $u_i \subseteq^* w$ for some $i \in \{1, ..., n\}$ can be simplified into $false$.

4 Multiset Patterns as Predictors

In this section we propose a methodology based on multiset patterns to compute a sufficient condition for the presence of a molecule s after k evolution steps of a given P system. The sufficient condition will be expressed as a pattern (called *predictor*) to be satisfied by the initial multiset w of the P system.

The idea is to define an operator that computes the predictor by starting from the pattern $(s, \{\})$ and by rewriting it by taking the set of rules of the P system into account. The pattern will be rewritten k times, each time simulating (in an abstract way) a backward step in the evolution of the P system. At each step, for each rule that is assumed to be applied the resulting pattern will include information of the rules competing with it for application.

The definition of the operator that computes a predictor for a molecule s in k steps is quite complex. We start with the definition of a few auxiliary functions and sets. Then, we choose to introduce the concepts by giving several examples of incremental complexity. Examples will be alternated with definitions of functions that formalize the introduced concepts. The complete definition of the operator, together with the related theoretical results, will conclude this section.

4.1 Auxiliary Functions and Sets

Function *AppRules* gives the set of all the minimal multisets of rules necessary to produce v.

Definition 4. *Given a multiset $v \in V^*$ and a set of rules $R \in \wp(\mathcal{R})$, we define the function $AppRules : V^* \times \wp(\mathcal{R}) \to \wp(\mathcal{R}^*)$ as*

$$AppRules(v, R) = \{M \in R^* \mid v \subseteq^* \textstyle\bigcup^*_{r \in supp(M)} prod(r)^{|M|_r} \text{ and } \nexists M' \subseteq^* M \text{ s.t. } v \subseteq^* \bigcup^*_{r \in supp(M')} prod(r)^{|M'|_r}\}$$

Example 1. Consider the P system $\Pi = (\{A, B, C, D, E\}, w, R)$ where evolution rules of the set R are:

$$\begin{array}{ll} r_0 : AB \to C & r_2 : C \to AC \\ r_1 : BD \to C & r_3 : E \to A \end{array}$$

We have: $AppRules(CCA, R) = \{r_0r_0r_3, r_0r_1r_3, r_1r_1r_3, r_0r_2, r_1r_2, r_2r_2\}$.

In order to simulate a backward step in the evolution of a P system we have to take into account that a molecule might be obtained as the product of an applied evolution rule, but also might be obtained since it was present in the previous step and left unchaged. This is not possible if there is a rule in the P system having such a molecule as the only reactant. As a consequence, in order to simulate the backward step of a P system, we consider an extended set of evolution rules that includes also *self rules* rewriting each molecule into itself, for each molecule that is not the only reactant of a rule of the P system.

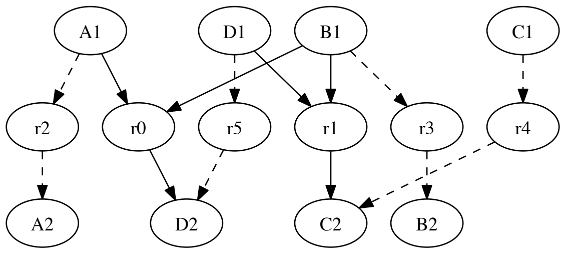


Fig. 1. Rules $R_1 \cup \mathit{Self}_{R_1}$

Definition 5. *Given a set of rules $R \subseteq \mathcal{R}$, the set of* self rules Self_R *is defined as*

$$\mathit{Self}_R = \{v \to v \mid |v| = 1 \text{ and } \forall r \in R,\ \mathit{react}(r) \neq v\}$$

Example 2. For the P system of Example 1 we have

$$\mathit{Self}_R = \{r_4 : A \to A, r_5 : B \to B, r_6 : D \to D\}$$

4.2 Competition for Reactants

Example 3. Consider the P system $\Pi_1 = (\{A, B, C, D\}, w, R_1)$ where the evolution rules R_1 are:

$$r_0 : AB \to D$$
$$r_1 : BD \to C$$

A visual representation of evolution rules in $R_1 \cup \mathit{Self}_{R_1}$ is given by the graph in Fig 1. The nodes in the top of the graph represent molecules used as reactants (associated with index 1), while the nodes in the bottom of the graph represent products (associated with index 2). Reactions are represented as nodes in the middle of the graph and by the arcs connecting such nodes to reactants and products. Solid arcs represent the evolution rules in R_1, while dashed arcs the evolution rules in $\mathit{Self}_{R_1} = \{r_2, r_3, r_4, r_5\}$.

The graph representation helps us to reason on backward steps of the P system. For instance, in order to obtain D in one step, we need to have either A and B, or D itself in the previous step. Moreover, the graph makes explicit the competition of evolution rules on common reactants. For example, the production of D by rule r_0 competes with the application of rule r_1 since both rules have B as a reactant. Similarly, the self rule $r_5 : D \to D$ competes with rule r_1. Note however that the contrary does not hold, indeed rule r_1 does not compete with the self rule $r_5 : D \to D$ because self rules represent molecules which are not consumed by the evolutions steps of rule in R_1.

The information on evolution rules competition is essential. In order to be sure that D is present after one step we need to be sure that either r_0 has been applied or D was already present and it has not been consumed by any other evolution rule not producing D.

We now define the function $competitor_1$, which results in the set of evolution rules competing for reactants with a given evolution rule r which produces a molecule of interest s.

Definition 6. *Given a rule $r \in \mathcal{R}$, a set of rules $R \subseteq \mathcal{R}$ and a molecule $s \in V$ such that $s \in supp(prod(r))$, we define*

$$competitor_1(r, R, s) = \{r' \in R \mid react(r) \cap^* react(r') \neq \emptyset \text{ and } s \notin prod(r')\}.$$

A pattern that characterizes a sufficient condition for the presence of D after one step can be easily obtained by combining the reactants of evolution rules producing D (including self rules) with the information on the reactants of the other evolution rules that compete with them and do not produce the molecule D.

For the case of Example 3, we can express a pattern that characterizes a sufficient condition for the production of D in one step as follows

$$\bigvee_{r \in AppRules(D, R_1 \cup Self_{R_1})} (react(r), react(competitor_1(r, R_1, D))$$

that corresponds to $(AB, \{BD\}) \vee (D, \{BD\})$. This pattern shows that D can be produced in two ways: through AB, that are the reactants of r_0, or through D itself. In both cases the only competitor is r_1, whose reactants are BD. So in both cases the pattern requires that AB or D remains after removing instances of BD in a maximal way. In other words, the pattern expresses the condition that either the multiset includes AB and the instances of B are more than the instances of D, or there is at least one D and the instances of D are more than the instances of B. Examples of multisets that satisfy the pattern (leading to the production of D) are ABB, AD, $ABBD$, etc.

Note that rule r_2 is not considered as a competitor since it is a self rule. Such a kind of rules cannot compete with other rules since they simply represent molecules that are not consumed by actual evolution rules of the P system.

In order to perform more than one backward step we will have to generalize the computation of the pattern representing the sufficient condition to the case in which we are interested in the production of *a multiset of molecules*, rather a single molecule. For instance, in the case of Example 3, performing one more backward step would require to compute the sufficient condition for the production of AB or of D, that will then be used to obtain D.

In order to show how to compute a pattern for the presence of a *multiset of molecules* after one step, consider, in the case of Example 3, the multiset DC. The pattern representing a sufficient condition for the presence of DC in one step could be obtained by combining the already seen pattern for the presence of D with analogous pattern for the presence of C, that is $(BD, \{AB\}) \vee (C, \{\})$. Since D and C are two different molecules (the case of repeated molecules is more complex and will be treated separately in Section 4.3) we can combine the two patterns by simply using a conjunction, thus obtaining:

$$((AB, \{BD\}) \vee (D, \{BD\})) \wedge ((BD, \{AB\}) \vee (C, \{\}))\,.$$

Multisets satisfying the pattern are DC, ABC, ADC, $ABBD$, etc. Indeed, such multisets allow us to obtain DC after one step according to the rules in R_1. On the other hand, multisets not satisfying the pattern are for instance ABD, DCB and $ABDC$. The latter, in particular, could lead to the production of DC (actually DDC), but also to the production of ACC that does not include DC.

Example 4. Consider now a P system $\Pi_2 = (\{A, B, C, D\}, w, R_2)$ where R_2 (depiceted in Fig 2) is the same as R_1 of Example 3, but with r_0 extended with one more product, namely

$$\begin{array}{l} r_0 : AB \to DC \\ r_1 : BD \to C \end{array}$$

While the sufficient conditions for molecule D to be present after one step are the same as in the previous example, the condition for the presence of C after one step has to take into account that now r_0 produces C, therefore it does not compete with r_1 for the production of C. This is correctly taken into account by the function $\mathit{competitor}_1$, indeed $\mathit{competitor}_1(r_0, R_2, C) = \emptyset$. Therefore, the pattern for the presence of C in one step, defined as

$$\bigvee_{r \in \mathit{AppRules}(C, R_2 \cup \mathit{Self}_{R_2})} (\mathit{react}(r), \mathit{react}(\mathit{competitor}_1(r, R_2, C)))\,,$$

turns out to be $(AB, \{\}) \vee (BD, \{\}) \vee (C, \{\})$.

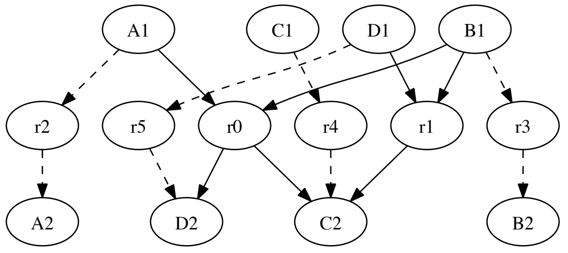


Fig. 2. Rules $R_2 \cup \mathit{Self}_{R_2}$

4.3 Competitors Dealing with Multiple Occurrences of Molecules

When multiple occurrences of the same molecule come into the picture, things get quickly more complicated.

Example 5. Consider the P system $\Pi_3 = (\{A, B, C, D\}, w, R_3)$ where the evolution rules $R_3 = \{r_0, r_1\}$, depicted in Fig. 3, are

$$\begin{aligned} r_0 &: AB \to D \\ r_1 &: BC \to D \end{aligned}$$

Assume we are interested in the multiset DD. To produce DD in the P system Π_3 we may either apply one rule twice, or the two rules together. The pattern for the presence of DD in one step cannot be obtained just as a conjunction of a pattern p expressing the sufficient condition for D with itself, since $p \wedge p$ is equivalent to p.

In order to deal with multiple occurrences of molecules we have to consider, in the computation of the backward step, the possible *multisets of evolution rules* that could have been applied in order to produce such molecules. These multisets of rules are given by the auxiliary function *AppRules* defined in Section 4.1.

At a first glance one may think of defining the pattern for the presence of DD in one step as follows:

$$\bigvee_{n \in AppRules(DD, R_3 \cup Self_{R_3})} \bigwedge_{r \in supp(n)} (react(r)^{|n|_r}, react(competitor_1(r, R_3, D)))$$

that would give the following result:

$$(ABAB, \{\}) \vee ((AB, \{\}) \wedge (BC, \{\})) \vee ((AB, \{\}) \wedge (D, \{\})) \\ \vee ((BC, \{\}) \wedge (D, \{\})) \vee (BCBC, \{\}) \vee (DD, \{\}))$$

This pattern is however not correct, since it is satisfied by multiset ABC (because $ABC \models (AB, \{\}) \wedge (BC, \{\})$) that does not lead to DD in one step.

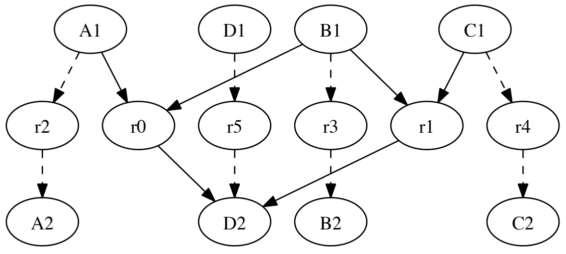


Fig. 3. Rules $R_3 \cup Self_{R_3}$

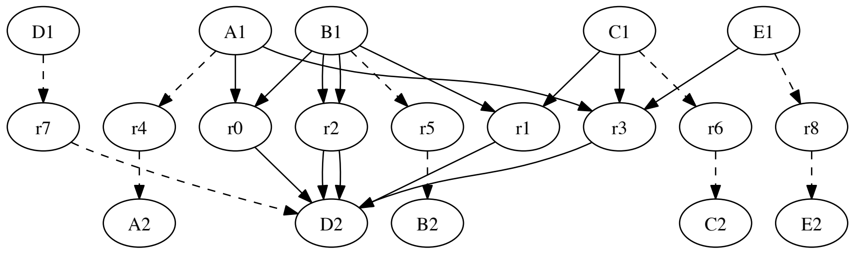


Fig. 4. Rules $R_4 \cup \mathit{Self}_{R_4}$

The point in this case is that there are two different rules that produce the same product D competing for the same reactant B. Since more than one instance of D has to be produced, we have to take also this form of competition into account. To this purpose, we define the function $\mathit{competitor}_2$.

Definition 7. *Given a rule $r \in \mathcal{R}$, a set of rules $R \subseteq \mathcal{R}$ and an multiset of rules $n \in R^*$, we define*

$$\mathit{competitor}_2(r, R, n) = \{r' \in R \mid r' \in \mathit{supp}(n), r' \neq r, \mathit{react}(r) \cap^* \mathit{react}(r') \neq \emptyset\}$$

Now, the pattern for DD can be expressed as

$$\bigvee_{n \in \mathit{AppRules}(DD, R_3 \cup \mathit{Self}_{R_3})} \bigwedge_{r \in \mathit{supp}(n)} (\mathit{react}(r)^{|n|_r}, \mathit{react}(C_{12}))$$

where $C_{12} = \mathit{competitor}_1(r, R_3, D) \cup \mathit{competitor}_2(r, R_3, n)$. The formula gives the following result:

$$(ABAB\{\}) \lor ((AB, \{BC\}) \land (BC, \{AB\})) \lor ((AB, \{\}) \land (D, \{\})) \\ \lor ((BC, \{\}) \land (D, \{\})) \lor (BCBC, \{\}) \lor (DD, \{\})$$

which now correctly models the required property.

4.4 Competition for Products

Example 6. Consider the P system $\Pi_4 = (\{A, B, C, D, E\}, w, R_4)$ where the evolution rules $R_4 = \{r_0, r_1, r_2, r_3\}$, depicted in Fig. 4, are

$$\begin{array}{ll} r_0 : AB \to D & r_2 : BB \to DD \\ r_1 : BC \to D & r_3 : ACE \to D \end{array}$$

Assume that, as in Example 5, we are interested in the presence of multiset DD after one step. The multiset DD can be produced by several combinations of rules in R_4. Rule r_2 has DD as product, but suffers from the competition of r_0 and of r_1 that, although producing the same kind of molecule, produce only one instance of such a molecule. Indeed, by starting from multisets ABB or BBC, we may obtain DD through r_2, but we may also obtain only one D, through r_0 or r_1, respectively.

Similarly, there are cases in which DD can be obtained by applying r_0 and r_1 toghether. Rule r_3, however, may compete with such a combination of rules, since in presence of E it may consume reactants necessary for the application of r_0 and r_1 giving only one D as a result.

This example suggests that the concept of competitor has to be enriched with a definition that takes into account when a rule competes with a multiset of rules $n \in R^*$ that produce more than one occurrence of a molecule. Intuitively, this occurs when the use of such a rule prevents the application of a subset of the rules in n without producing an equivalent number of occurrences of the required molecule. This form of competition is formalized by the function $\mathit{competitor}_3$ defined as follows.

Definition 8. *Given a rule $r \in \mathcal{R}$, a set of rules $R \subseteq \mathcal{R}$, a multiset of rules $n \in R^*$ and a molecule $s \in V$, we define:*

$$\begin{aligned}
&\mathit{competitor}_3(r, R, n, s) = \\
&\{r' \in R \mid s \in \mathit{prod}(r'), r' \notin \mathit{supp}(n), \mathit{react}(r) \cap \mathit{react}(r') \neq \emptyset, \\
&\qquad \exists m \subseteq^* n, \{r\} \subseteq^* m, \mathit{react}(m) \cap_* \mathit{react}(r') = \mathit{react}(n) \cap_* \mathit{react}(r'), \\
&\qquad \forall m' \subset^* m, \mathit{react}(m') \cap_* \mathit{react}(r') \neq \mathit{react}(n) \cap_* \mathit{react}(r'), \\
&\qquad |\mathit{prod}(m)|_s \supset^* |\mathit{prod}(r')|_s\}
\end{aligned}$$

Assume as before that we are interested in the production of DD in one step. The pattern expressing a sufficient conditions is then

$$\bigvee_{n \in \mathit{AppRules}(DD, R_3 \cup \mathit{Self}_{R_3})} \bigwedge_{r \in \mathit{supp}(n)} (\mathit{react}(r)^{|n|_r}, \mathit{react}(C_{123}))$$

where $C_{123} = \mathit{competitor}_1(r, R_3, D) \cup \mathit{competitor}_2(r, R_3, n) \cup \mathit{competitor}_3(r, R_3, n, D)$. The formula gives the following result:

$$\begin{aligned}
&((AB, \{BC, ACE\}) \wedge (BC, \{AB, ACE\})) \ \vee \ (BB, \{AB, BC\}) \ \vee \\
&\qquad ((AB, \{ACE, BC\}) \wedge (ACE, \{AB, BC\})) \ \vee \\
&\qquad\qquad ((BC, \{ACE, AB\}) \wedge (ACE, \{BC, AB\})) \ \vee \ (DD, \{\})
\end{aligned}$$

In the obtained pattern, conjunction $(AB, \{ACE, BC\}) \wedge (ACE, \{AB, BC\})$ is not satisfiable, since on the one hand it requires $ABACE$ to be included in the multiset, but at the same time it requires BC not to be included. The same holds for $(BC, \{ACE, AB\}) \wedge (ACE, \{BC, AB\})$ with $BCACE$ and AB. As a consequence, the pattern can be simplified into

$$((AB, \{BC, ACE\}) \wedge (BC, \{AB, ACE\})) \vee (BB, \{AB, BC\}) \ \vee \ (DD, \{\})$$

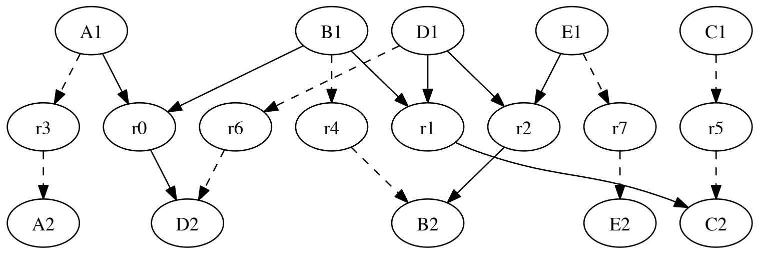


Fig. 5. Rules $R_5 \cup \mathit{Self}_{R_5}$

According to this pattern, for example, all multisets that contain BB lead to the presence of DD after one step as long as they contain enough instances of B (two more than the instances of A and C). As required, neither multiset ABB nor BBC satisfy the pattern.

This example shows also a situation in which the pattern does not describe multisets that actually lead to the presence of DD in one step, such as the multiset $ABBCE$. What happens in this case is that rule r_3 is identified as a competitor of both r_0 and r_1. However, in a multiset like $ABBCE$ the application of r_3 (that actually prevents r_0 and r_1 to be applied) causes also r_2 to be applied, obtaining DDD as result. This shows that the proposed notions of competitor are not able to characterize *all* multisets that lead to the presence of a required multiset in a given number of steps. In this case it is not able to recognize that the application of a competitor rule has as a side effect the application of some other rules that actually lead to the wanted result.

4.5 Multiple Backward Steps

We now describe how to obtain a pattern that expresses sufficient conditions for the presence of a molecule after two or more steps starting from patterns expressing sufficient conditions after one step.

Example 7. Let us consider the P system $\Pi_5 = (\{A, B, C, D, E, F\}, w, R_5)$ where the evolution rules in R_5, depicted in Fig. 5, are

$$r_0 : AB \to D$$
$$r_1 : BD \to C$$
$$r_2 : ED \to B$$

Note that rule r_0 and r_1 are the ones of Example 3 and the pattern for the presence of molecule D in one step is as in the previous example, namely $p = (AB, \{BD\}) \vee (D, \{BD\})$. In order to obtain the pattern expressing the sufficient condition for the presence of D after two steps, intuitively we have to consider all the ways a multiset satisfying p can be obtained in one step.

The pattern p is satisfied by multiset containing A and B, or D. Hence, we could compute the patterns that predict the presence of A, B and D in one step, and use them to construct a pattern for the satisfaction of p after one step. In addition to this, we have to pay attention to the competitors of A, B and D mentioned in p, namely the set $\{BD\}$. In order to construct the pattern for the satisfiability of p in one step we have to consider also all the ways the competitor BD can be obtained in one step.

We formally define an operator that considers all the possible ways a set of multisets representing competitors can be obtained in one step.

Definition 9. *Given a set of rules $R \subseteq \mathcal{R}$, a set of multiset $\{u_1, ..., u_n\}$ with each $u_i \in V^*$, we define*

$$Cr(R, \{u_1, ..., u_n\}) = \{react(n) \mid \begin{array}{l} n \in AppRules(u_i, R \cup Self_R)) \\ and\ u_i \in \{u_1, ..., u_n\}\} \end{array}$$

From the predictor of A in one step $(A, \{AB\})$, the predictor of B in one step $(ED, \{BD\}) \vee (B, \{AB, BD\})$, the predictor of D in one step $(AB, \{BD\}) \vee (D, \{BD\})$ and $Cr(R_5, \{BD\}) = \{BD, EDD, ABB, ABED\} = C_5$ we can construct a pattern that predicts the presence of D in two steps as follows:

$$((A, \{AB\} \cup C_5) \wedge (ED, \{BD\} \cup C_5)) \vee ((A, \{AB\} \cup C_5) \wedge (B, \{AB\} \cup C_5)) \\ \vee (AB, \{BD\} \cup C_5) \vee (D, \{BD\} \cup C_5)$$

The initial multiset AED satisfies the pattern, indeed, in one step we obtain AB using the only enabled rule r_2 and in two steps we obtain D applying the only enabled rule r_0. Consider $AEDD$ that does not satisfy the pattern, after one step we obtain ABD using the only enabled rule r_2 but also rule r_1 is enabled and by applying it we obtain AC that does not contain D.

4.6 Definition of the Main Operator and Theoretical Results

In the previous sections we have described the ingredients for the computation of a pattern expressing sufficient conditions for the presence of an molecule s after k steps. Now, we formally define an operator $\mathtt{Sc}_\Pi$ that performs such a computation.

Definition 10. *Let $\Pi = (V, w, R)$ be P system and $u \in V^*$. We define a function $\mathtt{Sc}_\Pi : V^* \times \mathbb{N} \to \mathcal{P}$ as follows:*

$$\mathtt{Sc}_\Pi(u, k) = \mathtt{Sca}_\Pi((u, \{\}), k)$$

where the auxiliary function $\mathtt{Sca}_\Pi : \mathcal{P} \times \mathbb{N} \to \mathcal{P}$ *is recursively defined as follows:*

$$\begin{aligned}
\mathtt{Sca}_\Pi(p,0) &= p\\
\mathtt{Sca}_\Pi(p_1 \vee p_2, k) &= \mathtt{Sca}_\Pi(p_1,k) \vee \mathtt{Sca}_\Pi(p_2,k)\\
\mathtt{Sca}_\Pi(p_1 \wedge p_2, k) &= \mathtt{Sca}_\Pi(p_1,k) \wedge \mathtt{Sca}_\Pi(p_2,k)\\
\mathtt{Sca}_\Pi((u,\{u_1,...,u_m\}),k) &= \mathtt{Sca}_\Pi\big(\bigwedge_{s\in supp(u)} p(s^{|u|_s},\{u_1,...,u_m\}),k-1\big)
\end{aligned}$$

where

$$p(s^i,U) = \bigvee_{n\in AppRules(s^i,R\cup Self_R)} \left(\bigwedge_{r\in supp(n)} (react(r)^{|n|_r}, \bigcup_{r'\in C_{123}} \{react(r')\} \cup Cr(R,U)) \right)$$

and

$$C_{123} = competitor_1(r,R,s) \cup competitor_2(r,R,n) \cup competitor_3(r,R,n,s)$$

Now we present some lemmata that, step by step, lead to the main theorem stating that the $\mathtt{Sc}_\Pi(u,k)$ operator actually computes a pattern representing a sufficient condition for the presence of u after k steps. In the lemmata and in the main theorem, given two multisets w and w' and a set of evolution rules R, we will denote with $w \to_R w'$ the fact that w' can be obtained from w by applying rules in R in a maximally parallel way.

The first lemma states that the portion of the pattern computed by the operator and defined as $p(s^i,U)$ in Def. 10 is a predictor for the presence of i instances of molecule s after one step of evolution of the P system.

Lemma 3. *Given a P system* $\Pi = (V, w_0, R)$*,* $w \in V^*$ *and* $s^i \in V^*$ *with* $s \in V$ *and* $i > 0$*, if* $w \models p(s^i,\emptyset)$ *then* $\forall w' \in V^*$ *such that* $w \to_R w'$*, it holds* $s^i \subseteq^* w'$*.*

The second lemma states that if a multiset w satisfies $p(s^i,U)$, then the multiset obtained after one evolution step will satisfy the basic pattern (s^i,U).

Lemma 4. *Given a P system* $\Pi = (V, w_0, R)$*,* $w \in V^*$ *and a basic pattern* (s^i,U) *with* $s \in V$ *and* $i > 0$*, if* $w \models p(s^i,U)$ *then* $\forall w' \in V^*$ *such that* $w \to_R w'$*, it holds* $w' \models (s^i,U)$*.*

Finally, the following lemma states that if a multiset w satisfies the pattern $\bigwedge_{s\in supp(u)} p(s^{|u|_s},U)$ with u a generic multiset, then the multiset obtained after one evolution step will satisfy the basic pattern (u,U).

Lemma 5. *Given a P system* $\Pi = (V, w_0, R)$*,* $u \in V^*$*,* $w \in V^*$ *and a basic pattern* (u,U) *with* $u \in V^*$*, if* $w \models \bigwedge_{s\in supp(u)} p(s^{|u|_s},U)$ *then* $\forall w' \in V^*$ *such that* $w \to_R w'$*, it holds* $w' \models (u,U)$*.*

We are now ready to state the main result of this paper.

Theorem 1 (Sufficient Condition). *Let* $\Pi = (V, w_0, R)$ *be P system and* $u \in V^*$*. If* $w_0 \models \mathtt{Sc}_\Pi(u,k)$ *then for any* $w_1,\ldots,w_k$ *such that* $w_{i\in[1,k]} \in V^*$ *and* $w_0 \to_R w_1 \to_R \ldots \to_R w_k$*, it holds* $u \subseteq^* w_k$*.*

The theorem essentially states that the pattern computed by the $\mathtt{Sc}_\Pi$ operator is actually a *sound* predictor.

5 Applications

5.1 P Systems as Language Acceptors

Let us consider again the example of the multiset language consisting only of the multiset A^3 we described in Section 3.2. An acceptor for such a language can be represented by the P system $\Pi_6 = (\{O, A, T, F\}, w_0, R_6)$, where R_6 consists of the following rules:

$$r_0 : OA^3 \to T$$
$$r_1 : TA \to F$$

The acceptor works by assuming that $|w_0|_O = 1$ and $|w_0|_T = |w_0|_F = 0$. If $|w_0|_A = 3$, then in one step T is produced and it is left unchanged in the second step (actually, the P system terminates after one step). If $|w_0|_A \neq 3$, then either T is not produced, or it is replaced by F in the second step. As a consequence, T will be present after two steps iff $|w_0|_A = 3$.

Let us compute the predictor of T in two steps for the P system Π_6 by applying the $\texttt{Sc}_{\Pi_6}$ operator:

$$\begin{aligned}
\texttt{Sc}_{\Pi_6}(T, 2) &= \texttt{Sca}_{\Pi_6}((T, \{\}), 2) = \texttt{Sca}_{\Pi_6}((OA^3, \{TA\}) \vee (T, \{TA\}), 1)\\
&= \texttt{Sca}_{\Pi_6}((OA^3, \{TA\}), 1) \vee \texttt{Sca}_{\Pi_6}((T, \{TA\}), 1)\\
&= \texttt{Sca}_{\Pi_6}((OA^3, \{TA, OA^4\}), 0) \vee \texttt{Sca}_{\Pi_6}((OA^3, \{TA, OA^4\}) \vee (T, \{TA\}), 0)\\
&= \texttt{Sca}_{\Pi_6}((OA^3, \{TA, OA^4\}), 0) \vee \texttt{Sca}_{\Pi_6}((OA^3, \{TA, OA^4\}), 0)\\
&\quad \vee \texttt{Sca}_{\Pi_6}((T, \{TA\}), 0)\\
&= (OA^3, \{TA, OA^4\}) \vee (OA^3, \{TA, OA^4\}) \vee (T, \{TA\})\\
&= (OA^3, \{TA, OA^4\}) \vee (T, \{TA\})\,.
\end{aligned}$$

Assumptions $|w_0|_O = 1$ and $|w_0|_T = 0$ allow us to simplify the obtained pattern into $(OA^3, \{OA^4\})$ that is exactly the pattern p'_5 we considered in Section 3.2.

We now consider an acceptor for the language A^nB^n. As in Section 3.2, we start by focusing on the complement of A^nB^n, namely A^nB^m with $n \neq m$. Let $\Pi_7 = (\{O, D, A, B, C, T, F\}, w_0, R_7)$ be a P system where rules in R_7 are:

$$\begin{array}{ll}
r_0 : AB \to C & r_3 : O \to D\\
r_1 : AD \to T & r_4 : FT \to T\\
r_2 : BD \to T &
\end{array}$$

If we assume that the initial multiset w_0 contains exactly one F, one O and no instances of T and of D, namely $|w_0|_F = |w_0|_O = 1$ and $|w_0|_T = |w_0|_D = 0$. Under such an assumption, the evolution of the P system is as follows: in the first step a maximal number of AB pairs are consumed by rule r_0 and, at the same time, molecule O is transformed into D by rule r_3. In the second step, if either some A or some B is still present, that is if the number of A was not the same as the number of B in the initial multiset, then one instance of T is produced by either r_1 or r_2. If T is produced, it causes F to be removed in the third step due to the application of rule r_4. As a consequence, after three steps molecule T is present iff the initial multiset contained different numbers of A and of B. Otherwise, molecule F is present instead of T.

Let us compute the predictor of T after three steps for the P system Π_7 by applying the $\mathtt{Sc}_{\Pi_7}$ operator:

$$\begin{aligned}
\mathtt{Sc}_{\Pi_7}(T,3) &= \mathtt{Sca}_{\Pi_7}((T,\{\}),3)\\
&= \mathtt{Sca}_{\Pi_7}((T,\{\}) \vee (FT,\{\}) \vee (BD,\{AB\}) \vee (AD,\{AB\}),2)\\
&= \mathtt{Sca}_{\Pi_7}((T,\{\}),2) \vee \mathtt{Sca}_{\Pi_7}((FT,\{\}),2)\\
&\quad \vee \mathtt{Sca}_{\Pi_7}((BD,\{AB\}),2) \vee \mathtt{Sca}_{\Pi_7}((AD,\{AB\}),2)\\
&= \mathtt{Sca}_{\Pi_7}((T,\{\}),1) \vee \mathtt{Sca}_{\Pi_7}((FT,\{\}),1)\\
&\quad \vee \mathtt{Sca}_{\Pi_7}((BD,\{AB\}),1) \vee \mathtt{Sca}_{\Pi_7}((AD,\{AB\}),1)\\
&\quad \vee \mathtt{Sca}_{\Pi_7}((BO,\{AB\}),1) \vee \mathtt{Sca}_{\Pi_7}((AO,\{AB\}),1)\\
&= \mathtt{Sca}_{\Pi_7}((T,\{\}),0) \vee \mathtt{Sca}_{\Pi_7}((FT,\{\}),0)\\
&\quad \vee \mathtt{Sca}_{\Pi_7}((BD,\{AB\}),0) \vee \mathtt{Sca}_{\Pi_7}((AD,\{AB\}),0)\\
&\quad \vee \mathtt{Sca}_{\Pi_7}((BO,\{AB\}),0) \vee \mathtt{Sca}_{\Pi_7}((AO,\{AB\}),0)\\
&= (T,\{\}) \vee (FT,\{\}) \vee (BD,\{AB\}) \vee (AD,\{AB\})\\
&\quad \vee (BO,\{AB\}) \vee (AO,\{AB\})\\
&= (T,\{\}) \vee (BD,\{AB\}) \vee (AD,\{AB\}) \vee (BO,\{AB\}) \vee (AO,\{AB\})\,.
\end{aligned}$$

The assumptions on the absence of T and D and on the presence of B in the initial multiset make the obtained pattern equivalent to $(B,\{AB\}) \vee (A,\{AB\})$, that is exactly the pattern we identified in Section 3.2 for A^nB^m with $n \neq m$,

For the same P systems Π_7, let us now compute the predictor for the presence of F after three steps. This actually should be a pattern characterizing A^nB^n (that we have seen in Section 3.2 cannot be expressed by the version of multiset patterns as they are introduced in this paper).

$$\begin{aligned}
\mathtt{Sc}_{\Pi_7}(F,3) &= \mathtt{Sca}_{\Pi_7}((F,\{\}),3) = \mathtt{Sca}_{\Pi_7}((F,\{FT\}),2)\\
&= \mathtt{Sca}_{\Pi_7}((F,\{FT,FAD,FBD,FFT\}),1)\\
&= (F,\{FT,FAD,FBD,FFT,FAO,FBO\})\,.
\end{aligned}$$

From the assumption on the absence of T and D in the initial multiset we have that the obtained pattern corresponds to $(F,\{FAO,FBO\})$. Moreover, from the assumpton on the presence of F and O we can conclude that the pattern is actually satisfied only when $|w_0|_A = |w_0|_B = 0$. Hence, the pattern is a correct predictor (since A^0B^0 belongs to the A^nB^n language), but it does not capture all the initial multisets that would lead to the presence of F in three steps.

The pattern obtained by the proposed operator represents a sufficient condition for the presence of some molecules after a given number of steps. The last example shows that there are cases in which a complete condition (without false negatives) cannot be expressed with the current definition of multiset patterns. However, the limited expressiveness of multiset patterns is not the only reason for the incompleteness of the $\mathtt{Sc}_{\Pi}$ operator. Indeed, there are also cases in which the operator fails in computing a complete pattern, even if the such a pattern could be expressed. We have shown this kind of situations in Example 6.

5.2 An Application to Gametogenesis

Lake frog (*Pelophylax ridibundus* Pallas, 1771) and pool frog (*Pelophylax lessonae* Camerano, 1882) can mate producing the hybrid edible frog (*Pelophylax esculentus* Linneus, 1758). The edible frog can coexist with one or both of the parental species giving rise to mixed populations. Usually the genotypes of *P. ridibundus*, *P. lessonae* and *P. esculentus* are indicated by RR, LL, and LR, respectively. In Europe there are mainly mixed populations containing *P. lessonae* and *P. esculentus*, called L-E systems. Hybrids in these populations reproduce in a particular way, called *hybridogenesis*. Hybridogenesis consists in a particular gametogenetic process in which the hybrids exclude one of their parental genomes, and transmit the other one, clonally, to eggs and sperm. This particular way of reproduction requires that hybrids live sympatrically with the parental species the genome of which is eliminated. In this way hybrids in a L-E system eliminate the L genome thus producing *P. esculentus* when mating with *P. lessonae*, and generating *P. ridibundus* when mating with other hybrids. Usually *P. ridibundus* generated in L-E complexes are inviable due to deleterious mutations accumulated in the clonally transmitted R genome. Because of inviability of *P. esculentus* × *P. esculentus* offspring, edible frog populations cannot survive alone, but they must act as a sexual parasite of the parental species *P. lessonae.* In L-E complexes, the reproductive pattern is the one described in the following table, where the subscribed y indicates the male sexual chromosome.

	LL	LR
L_yL	L_yL LL	L_yR LR
L_yR	LR	RR not viable

Note that the y chromosome, determining the sex of males, can occur only in the L genome, due to primary hybridization which involved, for size constraints, *P. lessonae* males and *P. ridibundus* females. The table shows that only one of the three possible matings resulting in viable offspring produce LL genotypes.

In [9,1,2] we studied the dynamics of frog populations by varying a set of biological parameters. Here we propose a simple example inspired by hybridogenesis in L-E complexes. We assume to have *P. esculentus* males and females, represented by $P_E^{\male}$ and $P_E^{\female}$, respectively, and *P. lessonae* males and females represented by $P_L^{\male}$ and $P_L^{\female}$. For the sake of simplicity we assume that each frog produces a single gamete. In particular, *P. esculentus* males produce $R^{\male}$ gametes, while *P. esculentus* females produce $R^{\female}$ gametes; *P. lessonae* males produce either $L^{\male}$ or $L_y^{\male}$ gametes, and *P. lessonae* females produce only $L^{\female}$ gametes. Gametes combine for producing the next generation of frogs.

The example is formalized by the P system $\Pi_{Frogs} = (V_{Frogs}, w_0, R_{Frogs})$ where V_{Frogs} contains all the molecules representing individual frogs and gametes, and the set R_{Frogs} consists of the following evolution rules:

$$\begin{array}{ccccc}
P_E^{\male} \to R^{\male} & P_E^{\female} \to R^{\female} & P_L^{\male} \to L^{\male} & P_L^{\male} \to L_y^{\male} & P_L^{\female} \to L^{\female}
\end{array}$$

$$\begin{array}{ccc}
L^{\male} L^{\female} \to P_L^{\female} & L^{\male} R^{\female} \to P_E^{\female} & R^{\male} L^{\female} \to P_E^{\female} \\
L_y^{\male} L^{\female} \to P_L^{\male} & L_y^{\male} R^{\female} \to P_E^{\male} & R^{\male} R^{\female} \to \epsilon
\end{array}$$

We want to check the possibility to obtain a particular frog in two steps. We assume to start from an initial configuration containing only frogs. This will allow us to discard all patterns which consider the possibility to have gametes in the initial situation.

First we check whether there is an initial configuration which gives the certainty to obtain a *P. esculentus* male, $P_E^{\male}$, in two steps. We have:

$$\begin{aligned}
\mathtt{Sc}_{\Pi_9}(P_E^{\male}, 2) &= \mathtt{Sca}_{\Pi_9}((P_E^{\male}, \{\}), 2) \\
&= \mathtt{Sca}_{\Pi_9}((L_y^{\male} R^{\female}, \{L_y^{\male} L^{\female},\ R^{\male} R^{\female}\}), 1) \\
&= \mathtt{Sca}_{\Pi_9}((P_L^{\male}, \{P_L^{\male},\ P_L^{\male} P_L^{\female}, L_y^{\male} L^{\female},\ R^{\male} R^{\female}, P_E^{\male} P_E^{\female}\}) \\
&\quad \wedge (P_E^{\female}, \{\ P_L^{\male} P_L^{\female}, L_y^{\male} L^{\female},\ R^{\male} R^{\female}, P_E^{\male} P_E^{\female}\}), 0) \\
&= false \wedge (P_E^{\female}, \{\ P_E^{\male} P_E^{\female}\}) = false
\end{aligned}$$

In the last step, the first operand of the $\wedge$ operator is always $false$. Thus there are no initial multiset of frogs which can ensure the production of a *P. esculentus* male. Recall that a *P. esculentus* male can be obtained only by a *P. lessonae* male producing the gamete $L_y^{\male}$. Unfortunately *P. lessonae* males could all produce the $L^{\male}$ gamete and no $L_y^{\male}$.

Let us consider the production of a *P. esculentus* female, $P_E^{\female}$, in two steps:

$$\begin{aligned}
\mathtt{Sc}_{\Pi_9}(P_E^{\female}, 2) &= \mathtt{Sca}_{\Pi_9}((P_E^{\female}, \{\}), 2) \\
&= \mathtt{Sca}_{\Pi_9}((L^{\male} R^{\female}, \{L^{\male} L^{\female},\ L_y^{\male} R^{\female}, R^{\male} R^{\female}\}) \\
&\quad \vee (R^{\male} L^{\female}, \{L_y^{\male} L^{\female},\ L^{\male} L^{\female}\ R^{\male} R^{\female}\}), 1) \\
&= \mathtt{Sca}_{\Pi_9}((L^{\male} R^{\female}, \{L^{\male} L^{\female},\ L_y^{\male} R^{\female}, R^{\male} R^{\female}\}), 1) \\
&\quad \vee \mathtt{Sca}_{\Pi_9}((R^{\male} L^{\female}, \{L_y^{\male} L^{\female}, L^{\male} L^{\female}, R^{\male} R^{\female}\}), 1) \\
&= \mathtt{Sca}_{\Pi_9}(((P_L^{\male}, \{P_L^{\male}, ...\}) \wedge (P_E^{\female}, \{\ ...\})), 0) \\
&\quad \vee (\mathtt{Sca}_{\Pi_9}((P_E^{\male}, \{P_L^{\male} P_L^{\female}, P_L^{\male} P_E^{\female}, P_E^{\male} P_E^{\female}, L_y^{\male} L^{\female}, L^{\male} L^{\female},\ R^{\male} R^{\female}\}), 0) \\
&\qquad \wedge \mathtt{Sca}_{\Pi_9}((P_L^{\female}, \{P_L^{\male} P_L^{\female}, P_L^{\male} P_E^{\female}, P_E^{\male} P_E^{\female}, L_y^{\male} L^{\female}, L^{\male} L^{\female},\ R^{\male} R^{\female}\}), 0)) \\
&= \ false \vee (P_E^{\male}, \{P_E^{\male} P_E^{\female}\}) \wedge (P_L^{\female}, \{P_L^{\male} P_L^{\female}\})
\end{aligned}$$

The pattern $(P_E^{\male}, \{P_E^{\male} P_E^{\female}\}) \wedge (P_L^{\female}, \{P_L^{\male} P_L^{\female}\})$ is satisfied by multisets containing both more *P. esculentus* males than *P. esculentus* females and more *P. lessonae* females than *P. lessonae* males. For example an initial multiset containing two *P. esculentus* males, one *P. esculentus* female and two *P. lessonae* females will produce at least one *P. esculentus* female in two steps.

6 Conclusions and Further Developments

In this paper we have defined multiset patterns. Such patterns were exploited to express sufficient conditions on initial multisets that ensure the presence a multiset of molecules after a given number of evolution steps. Necessary conditions could also be expressed with multiset patterns and computed by a very simple operator that simulates the application backward of rules without considering any competition between different rules. For example, in the case of the P systems Π_4 of Example 6 the necessary condition for the presence of DD after 1 step is expressed by the following pattern $(AABCE, \{\}) \vee (ABCCE, \{\}) \vee (AACCEE, \{\}) \vee (BB, \{\})$.

Further developments of our work include the investigation of multiset patterns under the viewpoint of the multiset languages they characterize. Moreover, extensions of multiset patterns could be studied in order to enrich their expressiveness, this would be useful also to allow a new notion of predictor to be proposed which satisfies the completeness property (absence of false negatives).

References

1. Barbuti, R., Bove, P., Milazzo, P., Pardini, G.: Minimal probabilistic P systems for modelling ecological systems. Theoretical Computer Science 608, 36–56 (2015)
2. Barbuti, R., Bove, P., Milazzo, P., Pardini, G.: Applications of p systems in population biology and ecology: The cases of mpp and app systems. In: International Conference on Membrane Computing. pp. 28–48. Springer (2016)
3. Barbuti, R., Gori, R., Levi, F., Milazzo, P.: Investigating dynamic causalities in reaction systems. Theoretical Computer Science 623, 114–145 (2016)
4. Barbuti, R., Gori, R., Levi, F., Milazzo, P.: Specialized predictor for reaction systems with context properties. Fundamenta Informaticae 147(2-3), 173–191 (2016)
5. Barbuti, R., Gori, R., Levi, F., Milazzo, P.: Generalized contexts for reaction systems: definition and study of dynamic causalities. Acta Informatica pp. 1–41 (2017), in Press.
6. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: Flat form preserving step-by-step behaviour. Fundamenta Informaticae 87, 1–34 (2008)
7. Bodei, C., Gori, R., Levi, F.: An analysis for causal properties of membrane interactions. Electr. Notes Theor. Comput. Sci. 299, 15–31 (2013)
8. Bodei, C., Gori, R., Levi, F.: Causal static analysis for brane calculi. Theor. Comput. Sci. 587, 73–103 (2015)
9. Bove, P., Milazzo, P., Barbuti, R.: The role of deleterious mutations in the stability of hybridogenetic water frog complexes. BMC evolutionary biology 14(1), 1 (2014)
10. Brijder, R., Ehrenfeucht, A., Main, M.G., Rozenberg, G.: A tour of reaction systems. Int. J. Found. Comput. Sci. 22(7), 1499–1517 (2011)
11. Brijder, R., Ehrenfeucht, A., Rozenberg, G.: A Note on causalities in reaction systems. ECEASST 30 (2010)
12. Busi, N.: Causality in membrane systems. In: Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers. pp. 160–171 (2007)
13. Ehrenfeucht, A., Rozenberg, G.: Reaction systems. Fundamenta informaticae 75(1-4), 263–280 (2007)

14. Gori, R., Levi, F.: Abstract interpretation based verification of temporal properties for bioambients. Inf. Comput. 208(8), 869–921 (2010)
15. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61, 108–143 (2000)

Universality of SNQ P Systems Using One Type of Spikes

Florin-Daniel Bîlbîe[1] and Andrei Păun[12]

[1] University of Bucharest, 14 Academiei St. District 1, 010014 - Bucharest,
`florin-daniel.bilbie@my.fmi.unibuc.ro`
`apaun@fmi.unibuc.ro`
[2] National Institute of Research and Development for Biological Sciences, 296 Independenței Bd. District 6, 060031 - Bucharest

Abstract. We investigate the recently defined spiking neural P systems with communication on request that are devices in the area of P systems abstracting the way the neurons work and process information. We are able to improve a recent result where the universality of the spiking neural P systems with communication on request was achieved with two types of spikes, in the current work only one type of spikes is sufficient for reaching the power of Turing Machines for these devices. Aside of the universality result, the paper defines a new manner in which the rule application for the request machines that is localizing the checks for each rule and making the implementation of the devices more feasible.

Keywords: P systems, universality, descriptional complexity, register machines, Spiking Neural Networks.

1 Introduction

This paper is a continuation of the results reported in [9] that introduced a new way of modeling for the Spiking Neural P systems (in short, SN P systems). Since the overall area is in the SN P systems, we will first give some details of these devices that were recently introduced in [3], and then investigated in [11] and [12], thus incorporating in membrane computing [10] ideas from spiking neurons, see, e.g., [1], [5], [6].

In short, an SN P system consists of a set of neurons placed in the nodes of a graph, representing synapses. The neurons send signals (spikes) along synapses (edges of the graph). This is done by means of firing rules, which are of the form $E/a^c \to a; d$, where E is a regular expression, c is the number of spikes consumed by the rule, and d is the delay from firing the rule and emitting the spike. The rule can be used only if the number of spikes collected by the neuron is “covered” by expression E, in the sense that the current number of spikes in the neuron, n, is such that $a^n \in L(E)$, where $L(E)$ is the language described by expression E.

The main modeling change from SN P systems from [9] is that a neuron will no longer “push” spikes along its axon to all its (inter)connected neurons, but request spikes from the neurons with whom it has synapses. The motivation behind this change comes from the Computer Science: to investigate the properties of such a change in the behavior of the device, but also from a more Philosophical view of computing devices that abstract the way the human brain works and then using said devices for solving pattern matching or classification problems (Artificial Neural Networks) as well as Biology where networks of heavily interconnected networks of neurons achieve a manner of sequentiality with or without a global clock.

As opposed to the SN P system model described above, the spiking neural P systems with communication on request will have rules in each neuron that will again be restricted by a regular expression E similarly as in the case of the SNP systems: if the number of spikes in a neuron is in the set of numbers defined by the regular expression E then that rule is applicable. The rest of the rule will specify neuron labels and numbers of spikes that are requested from each such neuron. We recall the main result from [9], namely universality of these devices but using two types of spikes in the construction. We show that we can improve this by reaching universality of the system with only one spike type with a minor change in the rule application: in [9] it is specified that if a rule requests spikes from i neurons, but one or more of these neurons does not have the requested number to be sent, then the whole rule will be blocked – will not execute. In this paper we consider that the regular expression is the only restriction on the applicability of the rules and if one of the neurons from the i neurons that receive the requests does not have the requested number, then the rest $i-1$ neurons will satisfy the request and only that neuron will deny the request locally. This means that the rule will execute but with the subset of the neurons that can satisfy the requests themselves. We consider this extension to be both reasonable and better than the original definition from the point of view of the implementation: the rule application check will happen locally in each neuron (whether the regular expression is satisfied by the current number of spikes) and then the requests will be sent on the axon and the spikes will be received only if the respective neurons can fulfill those requests. Furthermore, the case when conflicting requests will be received by a neuron can be also handled locally by that neuron that refuses to spike any value due to two or more conflicting requests received. We will provide in the following section more details about the rule application strategy in a formal manner.

In the following we give the formal definition of the P systems considered and basic notions and results on register machines in Sect. 2 that will be used in Sect. 3 that gives the main result of the paper. Finally, in Sect. 4 we give the conclusions and ideas for future extensions of this work.

2 Previous results and definitions

In this section we recall the formal definition of spiking neural P systems with communication request and notions from computability theory that will be useful in the rest of the paper.

2.1 Spiking neural P systems with communication on request

In the following we are recalling the definition of spiking neural P systems with communication on request from [9]. A spiking neural P system with communication on request (SNQ P system) with k types of spikes is a construct of form:

$$\Pi = (O, \sigma_1, \cdots, \sigma_m, a_{i_0}, out)$$

where:

1. $O = \{a_1, a_2, \cdots, a_k\}$ is an alphabet (a_i is called a type of spike), $k \geq 1$.
2. $\sigma_1, \cdots, \sigma_m$ are neurons, of the form $\sigma_i = (a_1^{n_1} a_2^{n_2} \cdots a_k^{n_k}, R_i)$, $1 \leq i \leq m$, $n_t \geq 0$, $1 \leq t \leq k$, where
   (a) n_t is the initial number of spikes of type a_t, contained by σ_i
   (b) R_i is a finite set of rule of form E/Qw, with E a regular expression over O and w a finite non-empty list of queries of forms (a_s^p, j) and (a_s^∞, j), $1 \leq s \leq k$, $p \geq 0$, $1 \leq j \leq m$
3. a_{i_0}, $1 \leq i_0 \leq k$ is the output spike and $out \in \{1, 2, \cdots, m\}$ indicates the output neuron.

A query of form (a_s^p, j) means that neuron σ_i is requesting p copies of a_s from σ_j, while a query of form (a_s^∞, j) means that σ_i requests all spikes of type a_s, no mater how many they are.

A rule E/Qw can be applied when the content of the neuron is described by the regular expression E and if at least one query in w can be executed (the query (a^p, j) cannot be satisfied if σ_j has less than p spikes). In [9] rule applicability was restricted by both regular expression E and the satisfiability of every single query. We believe that this relaxation of the rule application – the request is sent to a number of neurons and only the ones that can satisfy the request will execute – is more natural and probably easier to implement. With this relaxation of the application of the rules we are able to build an universal SNQ P system using a single type of spikes as usual in the Spiking Neuron Systems area (despite [9] where the universality was achieved with two types of spikes).

Another case when a rule cannot be applied is conflicting queries. If a neuron i_1 ask for u spikes from neuron j and at the same time neuron i_2 asks for v spikes from neuron j, with $u \neq v$, then the queries (a^u, j) and (a^v, j) are in conflict. In this case only one of them will be used, nondeterministically chosen.

At each step of computation we have 3 sub-steps (detailed in [9]):

- Each neuron chooses which rule will apply, if any.
- Requested spikes are removed from "source" neurons. If two or more (lets say y) queries ask for the same amount (say x) of the same spike, then the amount of spikes (y) is multiplied as many times (y times) as it is necessary to satisfy all queries. So, in the end, the source neurons only loses x spikes not $x \cdot y$ spikes
- Requested spikes are delivered to "destination" neurons.

2.2 Register machines

Using the model from above we will show that such systems are capable of universal computation. To do this we will prove that the Spiking P systems with communication on request are able to simulate correctly the work of any register machine. In the following we give the definition of Register Machines and the main results (of universality of Register Machines) that will be used in the main theorem of the paper.

In the proofs from the next sections we will use register machines as devices characterizing NRE, hence the Turing computability.

Informally speaking, a register machine consists of a specified number of registers (counters) which can hold any natural number, and which are handled according to a program consisting of labeled instructions; the registers can be increased or decreased by 1 – the decreasing being possible only if a register holds a number greater than or equal to 1 (we say that it is non-empty) –, and checked whether they are non-empty.

Formally, a (non-deterministic) *register machine* is a device M of form $M = (m, B, l_0, l_h, R)$, where $m \geq 1$ is the number of counters, B is the (finite) set of instruction labels, l_0 is the initial label, l_h is the halting label, and R is the finite set of instructions labeled (hence uniquely identified) by elements from B (R is also called the *program* of the machine). The labeled instructions are of the following forms:

- $l_1 : (\mathtt{ADD}(r), l_2, l_3)$, $1 \leq r \leq m$ (add 1 to register r and go non-deterministically to one of the instructions with labels l_2, l_3),
- $l_1 : (\mathtt{SUB}(r), l_2, l_3)$, $1 \leq r \leq m$ (if register r is not empty, then subtract 1 from it and go to the instruction with label l_2, otherwise go to the instruction with label l_3),
- $l_h : \mathtt{HALT}$ (the halt instruction, which can only have the label l_h).

A register machine generates a natural number in the following manner: we start computing with all m registers being empty, with the instruction labeled by l_0; if the computation reaches the instruction $l_h : \mathtt{HALT}$ (we say that it halts), then the values of register 1 is the number generated by the computation. The set of numbers computed by M in this way is denoted by $N(M)$. It is known (see [7]) that non-deterministic register machines generate exactly the family NRE, of Turing computable sets of numbers. Further, register machines with only three registers one of them being non-decreasing are universal.

We refer the interested reader to [7] for the universality proof of the register machines and to [13] p.2 as well as [7] p.170-174 for the universality of register machines with 3 registers.

Moreover, without loss of generality, we may assume that in the halting configuration all registers except the third one, where the result of the computation is stored are empty and the third register is non-decreasing.

3 Universal SNQ P System using one type of spike

In [9] it was proven that universality is achieved for SNQ P systems with two types of spikes, by simulating register machines, which are known to be equivalent with Turing machines. Formally, let $NSN_kP_m(Q, s)$ denote the set of number generated by SNQ P systems using at most k types of spikes and m neurons and working with strategy $s \in \{restrictive,\ localized\}$ where the *restrictive* strategy is the originally defined strategy where if one neuron from the list of the requests of the rule cannot satisfy the request, the entire rule is not applicable, and *localized* is the one that is localizing the rule checking and if one of the i requests cannot be satisfied, then the rule is applied, but that neuron will not fire. If k or m can be arbitrary, the we replace the corresponding value with $*$. In [9] was shown that $NSN_2P_*(Q,\ restrictive) = NRE$. In this paper we prove that we can achieve universality using one type of spike (i.e. the following theorem).

Theorem 1. $NSN_1P_*(Q,\ localized) = NRE$.

Proof. We are going to build a SNQ P system using a single type of spike denoted by a, which will also be the output spike. Our system will have the following neurons: label neurons l_i, helper neuron l'_i, for $1 \leq i \leq t$ (where t is the number of instructions of register machine), register neurons r_j, for $1 \leq j \leq s$ (where s is the number of registers used by the register machine), c_1, c_2, c_3, halt neuron l_h, halt helper neurons l'_h and l''_h, rep and l_{out} (the output neuron).

Each instruction label neuron will start with four spikes except neuron l_0 (the first instruction) which will start with 2 spikes (or it can start with four and will have a neuron which will ask for two spikes from neuron l_0, no mater its the content).

We encode the value n in register R by having $4(n + 1)$ spikes in neuron r (if R is supposed to have the value 0 then r will have 4 spikes). The instruction l_i will be active when two spikes will be removed from neuron l_i. To add new spikes into the system we are going to use a query of the form (a, env).

ADD module We are going to execute the instruction $l_i : (ADD(R), l_k, l_j)$ by constructing a module which will simulate it. The ADD module will have five neurons: l_i(neuron which denotes label of the current instruction), r(neuron which will simulate the register), l'_i(a neuron which will help us to restore the correct number of spikes in l_i), l_j and l_k(neurons which denotes the labels of instructions which will follow l_i). The ADD module is depicted in Fig. 1.

As previously mentioned, the execution of instruction l_i will be triggered by removing from neuron l_i two spikes. Left with only two spikes, neuron l_i can use the first rule, $a^2/Q(a, r)$, which requests one spike from neuron r. At next step, two neurons, l_i and r, can apply rules. By applying its rule, neuron r, now with $4n + 3$ spikes left, request five spikes from the environment. Meanwhile neuron l_i apply its second rule, $a^3/Q(a^\infty, l'_i)$, and requests all spikes of neuron l'_i. At step two, neuron r has $4n + 8 = 4(n + 2)$ spikes, which is the equivalent of

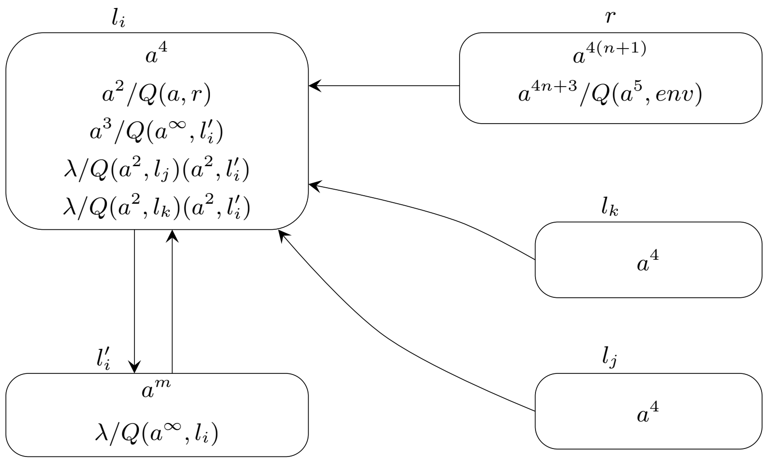


Fig. 1. The ADD module, simplified form.

adding 1 to register R and neuron l'_i is using its rule and request all spikes from l_i. Next, at step 3, l_i will use one of the two rules triggered in the absence of spikes, nondeterministically chosen. Depending on which rule was picked, either l_k or l_j will be activated and l_i will restore its initial number of spikes. Note that the initial number of spike in l'_i does not matter as long as it is greater than 1 (that's why in Fig. 1 we have m spikes in l'_i, with m arbitrary), because we just need some spikes to have the neuron inactive and by removing them we will have neuron l'_i active.

SUB module Next instruction which we simulate is $l_i : (SUB(R), l_k, l_j)$. Like ADD module, the SUB module will have five neurons: l_i(current instruction label), l_k (instruction l_k), l_j (instruction l_j), l'_i(helper neuron) and r(the register). The initial state of the system is depicted in Fig. 2. Label neurons l_i,l_k and l_j start with four spikes each, neuron r with $4(n+1)$ spikes and the helper neuron l'_i with $6u$(actually neuron l'_i will start with only 6 spikes, and with every successful subtraction made by l_i, l'_i will acquire 6 more spikes).

Activated by removing two spikes from it, neuron l_i will try to request 6 spikes from neuron r. Using its rule, $a^2/Q(a^6,r)(a,l'_i)$, neuron l_i requests 6 spikes from neuron r and one spike from neuron l'_i (this spike will be used to check if the request to r was satisfied or not). At step two we have two possibilities: if neuron r was storing a value bigger than 0 ($4(n+1)$ with $n > 0$) or neuron r was storing 0 (four spikes). Lets tackle the second case. If R was empty(storing 0) then neuron r has four spikes, meaning that request from l_i cannot be satisfied. At step two l_i has three spikes (two from before and one from l'_i) which trigger the second rule, $a^3/Q(a,l'_i)$ and acquire one more spike from l'_i. At step tree, neuron l'_i, after having two spikes removed, requests all spikes from l_i, storing now $6u+4$. With

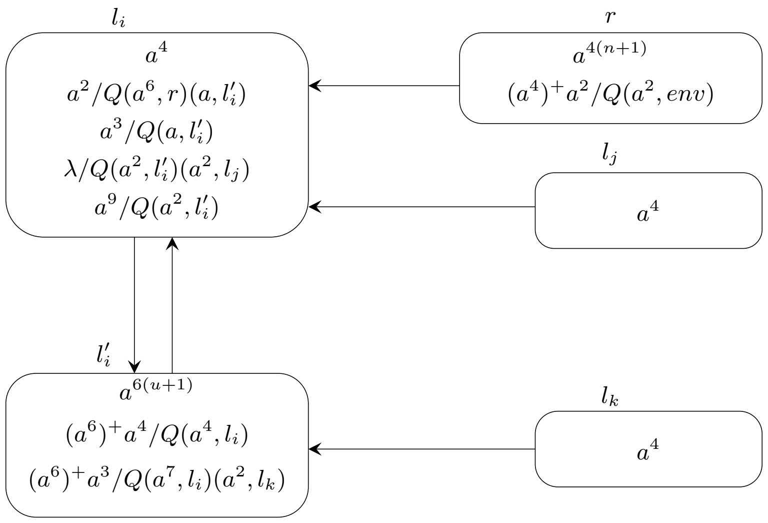


Fig. 2. The SUB module, simplified form.

no spikes left, neuron l_i requests two spikes from l_i' and two spikes from l_j. In the end, l_i restore its four neurons, l_i' has its $6u$ neurons back and l_j is activated.

In the second case, r has $4(n + 1)$ spikes, with $n \geq 1$. The query (a^6, r) is satisfied, r loses 6 spikes and l_i gains 6. In the second step, both neurons r and l_i can apply a rule. With 9 spikes, l_i requests two more from l_i', acquiring 11 spikes. Meanwhile r has $4n - 2$ and apply its rule and request two more from the environment, which makes a grand total of $4n$ spikes, the equivalent of substracting 1 from R (we started with $4(n + 1)$ spikes). In the third step, l_i' has $6u + 3$ spike and can trigger its second rule, $(a^6)^+a^3/Q(a^7, l_i)(a^2, l_k)$, and requests seven spikes from l_i and two from l_k, nine spikes in total. At the end, l_i has four spikes (initial configuration), l_i' has $6(u + 1)$ spikes (six more spikes) and l_k is the active instruction.

HALT module Because we encode the value n with $4(n + 1)$ spikes in register neurons, we have to construct a module which will output the value computed by the system. We will transform the $4n + 4$ spikes from the output register r of the register machine to n spikes in neuron l_{out}, which we designed as the output neuron of the SNQ P system.

The initial configuration of HALT module presented in Fig. 4 will become active when two spike are removed from neuron l_h (meaning that the register machine has reached the halt instruction l_h). Neuron l_h will request eight spikes the output register r and one spike from environment. We have two cases. If register r was empty (it stores the value 0 encoded as four spikes), then it cannot satisfy the query and l_h will receive only the spike from environment. With seven spikes l_h ask for all spikes from neuron *stop*, depicted in Fig. 3. The

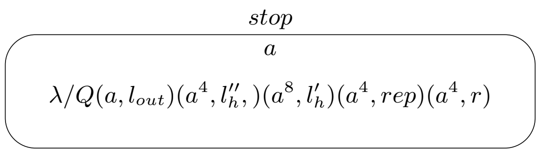

stop
a
$\lambda/Q(a, l_{out})(a^4, l''_h,)(a^8, l'_h)(a^4, rep)(a^4, r)$

Fig. 3. Stop neuron

role of neuron stop is to request the only spike from neuron l_{out} (in this way the value 0 computed by the register machine will be represented by 0 spikes in neuron l_{out}) and to stop any potential attempt of l_{out} to acquire other spikes, by removing all spikes from every other neuron from the halt module except l_h. Now, no rule cannot be applied and and the system stops.

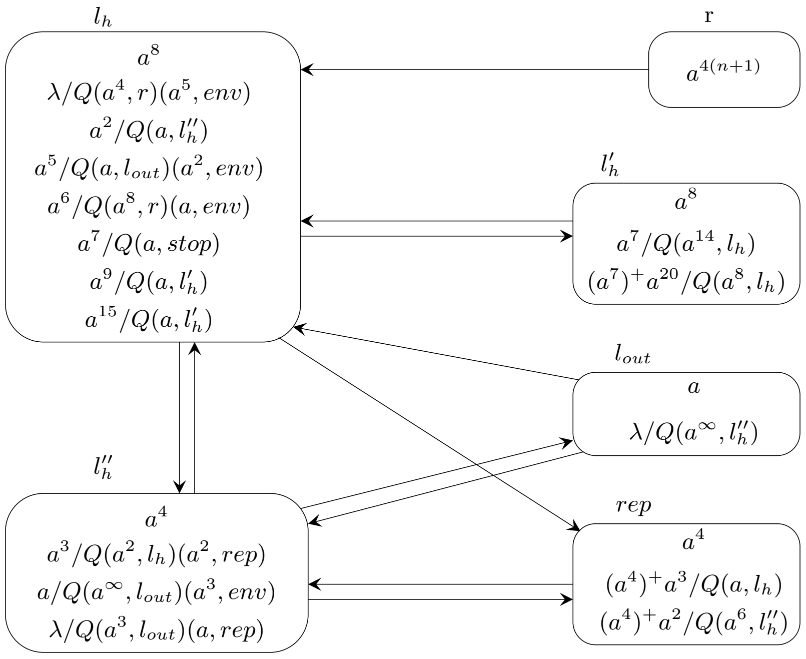
In the second case, neuron r is storing eight or more spike (register r was not empty), then l_h has its query satisfied and accumulates 15 spikes. Now we want to remove 13 spikes from l_h. First we request one spike from l'_h to l_h. With seven spikes left, l'_h will ask four 14 spikes from l_h (13 that we wanted to be removed from l_h and one that we previously requested). At fourth step, l_h ask for one spike from l''_h. This query will start a flow which will end with incrementation by one of the number of spikes from l_{out}. At fifth step, l''_h will remove two spikes from l_h by requesting them along with two spikes from rep, due to having its first rule, enabled by having three spikes. Next, neuron rep, with 2 spikes, will remove the extra six spikes from l''_h using its second rule activated by having $4p + 2$, with $p \geq 0$.

With one spike l''_h will request all spikes from neuron l_{out} and three spikes from environment. (we need those three spikes from neuron rep to be sure that by retrieving all spikes from l_{out} we are not enabling any rule in l''_h) Being empty, l_{out} will ask back for all spikes in l''_h (the spikes that it initially had and four additional spikes of l''_h). At this moment l_{out} has with three spikes more than what we wanted (remember that we wanted to increase the number of spikes by one, but we increased with four).

At ninth step, it is the turn of l''_h to apply its final rule and request the three additional spikes of l_{out} and one from rep. l''_h restore its initial configuration, l_{out} has with one more spike than what it started. All that it is left is to remove the one spike from l_h, which is done by the first rule of rep.

At eleventh step, l_h is empty and tries to request four more spikes from r, along with five from the environment. If the query can be satisfied by r, the l_h accumulates 9 spikes. Next l_h request again one spike from l'_h, which responds by requesting eight spikes from l_h by the mean of its second rule enabled by the presence of $7m + 20$, with $m \geq 0$. With two spikes in l_h neuron will start to redo the steps to increment l_{out} with one more spike.

If r cannot satisfy the query of l_h, then we have to remove one spike from l_{out}, because it has with a spike more than the number encoded by r (the initial one). This is achieved by the third rule of l_h. This way, we remove one spike from l_{out} and request two more from environment to be sure that we are not creating the setup for another rule. Now the system stops because no rule are available (be removing one spike from l_{out} we are not leaving it empty because this rule is applied after at least one time l_{out} was incremented with one).



l_h
a^8
$\lambda/Q(a^4, r)(a^5, env)$
$a^2/Q(a, l''_h)$
$a^5/Q(a, l_{out})(a^2, env)$
$a^6/Q(a^8, r)(a, env)$
$a^7/Q(a, stop)$
$a^9/Q(a, l'_h)$
$a^{15}/Q(a, l'_h)$
r
$a^{4(n+1)}$
l'_h
a^8
$a^7/Q(a^{14}, l_h)$
$(a^7)^+a^{20}/Q(a^8, l_h)$
l_{out}
a
$\lambda/Q(a^\infty, l''_h)$
l''_h
a^4
$a^3/Q(a^2, l_h)(a^2, rep)$
$a/Q(a^\infty, l_{out})(a^3, env)$
$\lambda/Q(a^3, l_{out})(a, rep)$
rep
a^4
$(a^4)^+a^3/Q(a, l_h)$
$(a^4)^+a^2/Q(a^6, l''_h)$

Fig. 4. Halt Module

Spike generator module Depicted in Fig. 5, the spike generator module will help us to replace all queries to environment with proper queries within the SNQ P system. To use this module, we have to replace in all queries *env* with c_1.

When used to simulate ADD instructions, the neuron r will request five spikes from neuron c_1. With no spikes left, c_1 asks for five spikes to neurons c_2 and c_3 (all spikes in those two). Empty of spikes, c_2 and c_3 query c_1, which now has 10 spikes, for five spikes each. The query is satisfied and both c_2 and c_3 replenish their spikes. Neuron c_1, loses 5 spikes and remains with 5 spikes.

In the case of SUB instruction, r is requesting for two spikes, and c_1 remains with three spikes. Now, c_1 uses its second rule and requests for two spikes to neurons c_2 and c_3. With 3 spikes left, c_2 and c_3 ask c_1(7 spikes) for 2 spikes each. Neuron c_1 remains with 5 spikes and neurons c_2 and c_3 acquire 2 spikes each.

For the simulation of HALT instruction, the spike generator module will work in a similar way with SUB and ADD instructions. Observe that this module is working only because each request to c_1 comes after at least two computational

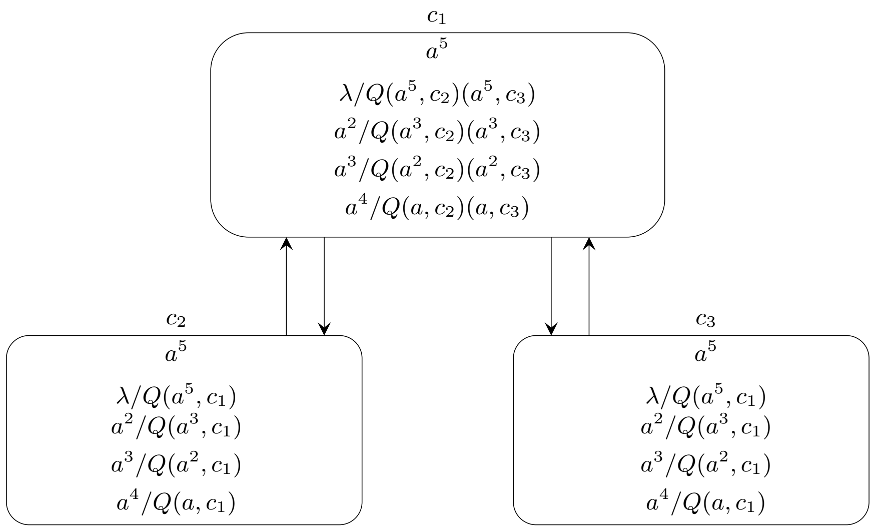


Fig. 5. Spike generator module

step after the previous. Otherwise this module will not work properly if it has to satisfy queries one after the other. $\square$

4 Conclusions and Future Research

We showed that the spiking neural P systems with communication on request are able to achieve universality using only one spike as is natural in this area (as spikes are encoded as electric current and it was shown that the amplitude of the spike does not encode information). We think that an investigation from the point of view of descriptional complexity would now be interesting: the obvious number of neurons in the system could be investigated as was done in other such constructs starting with the results of Korec [4] on small register machines, but also the number of synapses could be an interesting value that could be minimized, or even better, the size of the rules in the system (requests sent to a minimal number of neurons). In this moment our stop neuron has no less than eight other neurons where it requests spikes, without it the system is small: 2 requests at most per rule. Also, it remains to be seen if the universality with one type of spike can be achieved in the context described in [9] (use a rule when all requests are satisfiable). Finally, one could look at the max spiking and min-spiking sequentiality as defined in [2] that could lead to interesting results. We actually believe that min and max sequentiality can be achieved in this area and a further paper will provide those results.

References

1. W. Gerstner, W Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity.* Cambridge Univ. Press, 2002.
2. O. H. Ibarra, A. Păun, A. Rodrìguez-Patòn, Sequential SNP systems based on min/max spike number, *Theoretical Computer Science*, 410, (2009), 2982-2991.
3. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71 (2-3), (2006), 279–308.
4. I. Korec, Small universal register machines, *Theoretical Computer Science*, 168, (1996), 267–301.
5. W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8, 1, (2002), 32–36.
6. W. Maass, C. Bishop, eds.: *Pulsed Neural Networks*, MIT Press, Cambridge, 1999.
7. M. Minsky: *Computation – Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, NJ, 1967.
8. T. Neary: On the computational complexity of spiking neural P systems, *Natural Computing*, 9 (4), (2010), 831–851.
9. L. Pan, Gh. Păun, G. Zhang, SN P Systems with Communication on request, *Bulletin of IMCS Webpage*, (2017) 179–194.
10. Gh. Păun: *Membrane Computing – An Introduction.* Springer-Verlag, Berlin, 2002.
11. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *International Journal of Foundations of Computer Science*, 17 (04), (2006), 975–1002.
12. M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. manuscript, 2007.
13. R. Schroeppel: A two counter machine cannot calculate 2^N , Technical Report AIM-257, A. I. Laboratory, Massachusetts Institute of Technology, Cambridge, MA, May 1972, 1–31, available at ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-257.pdf (viewed Oct. 14, 2015)

APCol Systems with Teams

Lucie Ciencialová, Luděk Cienciala, and Erzsébet Csuhaj-Varjú

[1] Institute of Computer Science
and
Research Institute of the IT4Innovations Centre of Excellence,
Silesian University in Opava, Czech Republic
{lucie.ciencialova,ludek.cienciala}@fpf.slu.cz
[2] Department of Algorithms and Their Applications, Faculty of Informatics
ELTE Eötvös Loránd University, Budapest, Hungary,
Pázmány Péter sétány 1/c, 1117
csuhaj@inf.elte.hu

Abstract. In this paper, we investigate the possibility of “going beyond” Turing in the terms of Automaton-like P Colonies (APCol systems, for short), variants of P colonies processing strings as their environments. We use the notion of teams of agents as a restriction for the maximal parallelism of the computation. In addition, we assign a colour to each team. In the course of the computation, the colour is changing according to the team that is currently active. We show that we can simulate red-green counter machines with APCol systems with two-coloured teams of minimal size. Red-green counter machines are computing devices with infinite run on finite input that exceed the power of Turing machines.

Keywords: Automaton-like P colonies, APCol systems, red-green counter machine, unbounded computation, teams

1 Introduction

Recently, both unconventional Turing equivalent computing devices and computational models which “go beyond” Turing, i.e., which are able to compute more than recursively enumerable sets of strings or numbers are in the focus of interest. In membrane computing, we can find examples for both types of such constructs.

APCol systems (Automaton-like P colonies) were introduced in [5] as an extension of P colonies (introduced in [9]) - a very simple variant of membrane systems inspired by colonies of formal grammars. (The reader is referred to [14] for more information in membrane systems and to [10] and [7] for details on grammar systems theory.) An APCol system consists of a finite number of agents - finite collections of objects embedded in a membrane - and a shared environment. The agents are equipped with programs which are composed from rules that allow them to interact with their environment that is represented by a string. For this reason, the agents use their own objects and the objects of the environment. The number of objects inside each agent is set by definition and it is usually a very small number: 1, 2 or 3. The environmental string is processed by the agents and it is used as a communication channel for the agents as well. Through the string, the agents are able to affect the behaviour of another agent.

The activity of the agents is based on rules that can be rewriting, communication or checking rules [9]. A rewriting rule $a \to b$ allows the agent to rewrite (evolve) one object a to object b. Both objects are placed inside the agent. Communication rule $c \leftrightarrow d$ makes possible to exchange object c placed inside the agent with object d in the string. A checking rule is formed from two rules r_1, r_2 of type rewriting or communication. It sets a kind of priority between the two rules r_1 and r_2. The agent tries to apply the first rule and if it cannot be performed, then the agent executes the second rule. The rules are combined into programs in such a way that all objects inside the agent are affected by execution of the rules. Consequently, the number of rules in the program is the same as the number of objects inside the agent.

The interested reader can find more details on P colonies in [14], [8] and [4].

In this paper, we focus on APCol systems with agents forming teams; the concept was first proposed in [6]. The team is a finite number of agents of the APCol system. These collections can be so-called prescribed teams (given together with the components of the APCol system) or so-called free teams where only the size of the teams, i.e., the number of the agents in the team is given in advance. The notion is inspired by the concept of team grammar systems (see [15]). APCol systems with prescribed or with free teams function in the following manner: in every computation step only one team is allowed to work (only one team is active) and all of its components should perform a program in parallel.

Another interesting extension is to assign colours to programs, instructions or rules and observing how the currently used colour changes under the computation. This method is well-known for observing unbounded computations. Motivated by the notion of red-green Turing machines [12] (red-green register machines) and related notions in P systems theory [2], we introduce the concept of APCol systems with coloured teams. These constructs are APCol systems with teams where each team is associated with a colour. A string is accepted by an APCol system with coloured teams, if starting with the string as initial string the computation is unbounded and its teams with the final colour are active in an infinite number of steps and the teams of the other colours are active only in a finite number of steps.

Red-green Turing machines, introduced in [12] exceed the power of Turing machines since they recognize exactly the Σ_2-sets of the Arithmetical Hierarchy. These machines are deterministic and their state sets are divided into two disjoint sets, called the set of red states and the set of green states. Red-green Turing machines work on finite input words with the following recognition criterion on infinite runs: no red state is visited infinitely often and one or more green states are visited infinitely often. A change from a green state to a red state or reversely is called a mind change; we may speak of a change of the "colour". In [12], it is shown that every recursively enumerable language can be recognized by a red-green Turing machine with one mind change. It is also proved that if more than one mind changes may take place, then red-green Turing machines are able to recognize the complement of any recursively enumerable language.

Our paper is structured as follows: the second section is devoted to definitions and notations used in the paper. The third section contains results obtained on APCol systems with coloured teams, namely, we show that any red-green counter machine can be simulated with an APCol system with coloured teams, where there are two colours. The teams either consist of only one agent and then the system works sequentially, or the APCol system has teams of at most two agents acting in parallel. Finally, some conclusions are derived.

2 Definitions

Throughout the paper we assume that the reader is familiar with the basics of formal language and automata theory; for further details consult [15]. We list the notations used in the paper.

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.

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

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.

2.1 Register and Counter Machines

We briefly recall the basic notions, following the notations used in [2].

A register machine [11] is a construct $M = (m, B, l_0, l_h, P)$, where m is the number of registers, B is a set of labels, $l_0 \in B$ is the initial label, $l_h \in B$ is the final label, and P is the set of instructions bijectively labelled by elements of B. The instructions of M can be of the following forms:

- $l_1 : (ADD(r), l_2, l_3)$, with $l_1 \in (B - \{l_h\})$, $l_2, l_3 \in B, 1 \leq j \leq m$. It increases the value of register r by one and the next instruction to be performed is non-deterministically chosen, it is labelled by l_2 or l_3. This instruction is called increment.
- $l_1 : (SUB(r), l_2, l_3)$, with $l_1 \in (B - \{l_h\})$, $l_2, l_3 \in B, 1 \leq j \leq m$. If the value of register r is zero, then the label of the next instruction to be performed is l_3; otherwise, the value of register r is decreased by one and the label of

the next instruction to be executed is l_2. The first case is called zero-test, the second case is called decrement.
– $l_h : HALT$. The register machine stops executing instructions.

A configuration of a register machine is described by the numbers stored in the registers and by the label of the next instruction to be performed. Computations start by executing the instruction l_0 of P, and terminate by execution of the *HALT*-instruction l_h.

This model of register machines can be extended by instructions for reading from an input tape and writing to an output tape containing strings over an input alphabet T_{in} and an output alphabet T_{out}, respectively, see [2]:

– $l_1 : (read(a), l_2)$, with $l_1 \in (B - \{l_h\}), l_2 \in B, a \in T_{in}$. This instruction reads symbol a from the input tape and the next instruction is l_2.
– $l_1 : (write(a), l_2)$, with $l_1 \in (B - \{l_h\}), l_2 \in B, a \in T_{out}$. This instruction writes symbol a to the output tape and the next instruction is l_2.

This extended register machine, working with strings is also called a counter automaton and is denoted by $M = (m, B, l_0, l_h, P, T_{in}, T_{out})$. If no output is written, T_{out} is not indicated.

It is known (see e.g. [11]) that register machines with (at most) three registers can compute all recursively enumerable sets of natural numbers. Counter automata with two registers can simulate the computations of Turing machines and thus characterize RE. All these results are obtained with deterministic register machines, where the *ADD*-instructions are of the form $l_1 : (ADD(r), l_2)$, with $l_1 \in (B - \{l_h\}), l_2 \in B, 1 \leq j \leq m$. More details can be found in [2].

2.2 Red-Green Turing Machines

We briefly recall the most important notions and statements concerning red-green Turing machines and their variants, following [12], [1], and [2].

Red-green Turing machines, introduced in [12], exceed the power of the standard Turing machines, since they recognize exactly the Σ_2-sets of the Arithmetical Hierarchy. As we told before, they are deterministic and their state sets are divided into two disjoint sets, namely, the set of red states and the set of green states. Red-green Turing machines work on finite inputs with the recognition criterion on infinite runs that no red state is visited infinitely often and one or more green states are visited infinitely often. A change from a green state to a red state or reversely is called a mind change; we may speak of a change of the "colour". In [12], it was shown that every recursively enumerable language can be recognized by a red-green Turing machine with one mind change. It was also proved that if more than one mind change may take place, then they are able to recognize the complement of any recursively enumerable language.

In the analogy of the concept of red-green Turing machines, red-green counter machines (red-green register machines) were defined and examined [1]. The authors proved that the computations of a red-green Turing machine TM can be simulated by a red-green register machine RM with two registers and with string input in such a way that during the simulation of a transition of TM leading from a state p with colour c to a state p' with colour c' the simulating register machine uses instructions with labels (states) of colour c and only in the last step of the simulation changes the label (state) to colour c'. They showed that the reverse simulation works as well: the computations of a red-green register machine RM with an arbitrary number of registers and with string input can be simulated by a red-green Turing machine TM in such a way that during the simulation of a computation step of RM from an instruction with label (state) p with colour c to an instruction with label (state) p' with colour c', the simulating Turing machine TM are in states of colour c and only in the last step of the simulation changes to a state of colour c'.

In [2], the above notions were implemented for membrane systems: the notions of a red-green P automaton and its variants, as counterparts were introduced. It was shown that these devices are able to "go beyond" Turing, in the sense as red-green Turing machines are able to do.

2.3 APCol Systems

In the following we recall the concept of APCol systems, particular variants of P colonies, where the environment of the agents is given in the form of a string [5].

The agents of APCol systems contain objects, each object is an element of a finite alphabet. With every agent, a set of programs is associated. There are two types of rules in the programs. The first one is of the form $a \ \rightarrow b$ and it is called an evolution rule. It means that object a inside of the agent is rewritten (evolved) to object b. The second type of rules is called a communication rule and it is in the form $c \leftrightarrow d$. When this rule is performed, then the object c inside the agent and a symbol d in the string are exchanged. If $c = e$, then the agent erases d from the input string and if $d = e$, then the symbol c is inserted into the string.

During the work of the APCol system, the agents perform programs. The number of objects inside the agents remain unchanged during the functioning of the system, it is usually 2.

Since both rules in a program can be communication rules, an agent can work with two objects in the string in one step of the computation. In the case of program $\langle a \leftrightarrow b; c \leftrightarrow d\rangle$, a substring bd of the input string is replaced by string ac. If the program is of the form $\langle c \leftrightarrow d; a \leftrightarrow b\rangle$, then a substring db of the input string is replaced by string ca. That is, the agent can act only in one place in a computation step and the change of the string depends both on the order of the rules in the program and on the interacting objects. In particular, the following types of programs with two communication rules are considered:

- $\langle a \leftrightarrow b; c \leftrightarrow e\rangle$ - b in the string is replaced by ac,
- $\langle c \leftrightarrow e; a \leftrightarrow b\rangle$ - b in the string is replaced by ca,
- $\langle a \leftrightarrow e; c \leftrightarrow e\rangle$ - ac is inserted in a non-deterministically chosen place in the string,

– $\langle e \leftrightarrow b; e \leftrightarrow d\rangle$ - bd is erased from the string,
– $\langle e \leftrightarrow d; e \leftrightarrow b\rangle$ - db is erased from the string,
– $\langle e \leftrightarrow e; e \leftrightarrow d\rangle$; $\langle e \leftrightarrow e; c \leftrightarrow d\rangle$, ...- these programs can be replaced by programs of type $\langle e \rightarrow e; c \leftrightarrow d\rangle$.

The program is said to be *restricted* if it is formed from one rewriting and one communication rule. The APCol system is restricted if all of the programs of the agents are restricted.

To help the reader in the easier understanding the technical details of the paper, we recall the formal definition of an APCol system.

Definition 1. *[5] An APCol system is a construct*
$\Pi = (O, e, A_1, \ldots, A_n)$, where

– *O is an alphabet; its elements are called the objects,*
– *$e \in O$, called the basic object,*
– *A_i, $1 \le i \le n$, are agents. Each agent is a triplet $A_i = (\omega_i, P_i, F_i)$, where*
  - *ω_i is a multiset over O, describing the initial state (content) of the agent, $|\omega_i| = 2$,*
  - *$P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ is a finite set of programs associated with the agent, where each program is a pair of rules. Each rule is in one of the following forms:*
    - *$a \rightarrow b$, where $a, b \in O$, called an evolution rule,*
    - *$c \leftrightarrow d$, where $c, d \in O$, called a communication rule,*
  - *$F_i \subseteq O^*$ is a finite set of final states (contents) of agent A_i.*

At the beginning of the computation of the APCol system is in initial configuration which is an $(n+1)$-tuple $c = (\omega; \omega_1, \ldots, \omega_n)$ where ω is the initial state of the environment and the other n components are multisets of strings of objects, given in the form of strings, the initial states of the agents. The initial state of the environment does not contain object e.

A configuration of an APCol system Π is given by $(w; w_1, \ldots, w_n)$, where $|w_i| = 2$, $1 \le i \le n$, w_i represents all the objects placed inside the i-th agent and $w \in (O - \{e\})^*$ is the string to be processed.

In each computation step every agent attempts to find one of its programs to use. If it has applicable programs, then it non-deterministically chooses one of them and applies it. As usual in membrane computing, APCol systems work in the maximally parallel manner, i.e., as many agents perform one of its programs in parallel as possible. We note that other working modes can also be defined.

By applying programs, the APCol system passes from one configuration to another configuration. A sequence of configurations starting from the initial configuration is called a computation. A configuration is halting if the APCol system has no applicable program.

The result of computation depends on the mode in which the APCol system works. In the accepting mode, a string ω is accepted by APCol system Π if there exists a computation by Π such that it starts in the initial configuration $(\omega; \omega_1, \ldots, \omega_n)$ and ends by halting in a configuration $(\varepsilon; w_1, \ldots, w_n)$, where at least one of $w_i \in F_i$ for $1 \leq i \leq n$. In the generating mode, a string w_F is generated by Π if and only if there exists a computation starting in an initial configuration $(\varepsilon; \omega_1, \dots, \omega_n)$ and the computation ends by halting in the configuration $(w_F; w_1, \dots, w_n)$, where at least one of $w_i \in F_i$ for $1 \leq i \leq n$.

An APCol system Π can generate or accept a set of numbers as well, i.e., $|L(\Pi)|$.

In [5] the authors proved that the family of languages accepted by jumping finite automata (introduced in [13]) is properly included in the family of languages accepted by APCol systems with one agent, and it is proved that any recursively enumerable language can be obtained as a projection of a language accepted by an APCol system with two agents.

In [3] the authors proved that restricted APCol systems with two agents working in the generating mode determine $\mathbb{N}\cdot\mathsf{RE}$, while if the APCol systems have only a single agent, then only a proper subset of $\mathbb{N}\cdot\mathsf{RE}$ can be obtained.

2.4 APCol Systems with Coloured Teams of Agents

As a restriction of the computation process, we can introduce teams into the concept of APCol system, as proposed in [6]. The team is a finite set of agents. These teams can be prescribed teams (given together with the components of the APC0L system) or free teams where only the size of the teams, i.e., the number of agents in the team is given in advance. The notion is inspired by the concept of team grammar systems (see [15]). APCol systems with prescribed or with free teams work in the following manner: at any computation step only one team is allowed to work (only one team is active) and all of its components should perform a program in parallel.

One other extension of the concept of APCol system is associating "colour" to the agents or, if the APCol system is with teams, to the teams. The concept is inspired by red-green Turing machines; the idea was first presented in [6], given in an informal manner, using only two colours, red and green.

Definition 2. *An APCol system with coloured teams is a construct*
$$\Pi = (O, e, A_1, \dots, A_n, C, f, B, B_{colours}, B_{teams}),\ \textit{where}$$

- *O is an alphabet; its elements are called the objects,*
- *$e \in O$, called the basic object,*
- *A_i, $1 \leq i \leq n$, are agents. Each agent is a triplet $A_i = (\omega_i, P_i, F_i)$, where*
  - *ω_i is a multiset over O, describing the initial state (content) of the agent, $|\omega_i| = 2$,*
  - *$P_i = \{p_{i,1}, \dots, p_{i,k_i}\}$ is a finite set of programs associated with the agent, where each program is a pair of rules. Each rule is in one of the following forms:*
    - *$a \to b$, where $a, b \in O$, called an evolution rule,*
    - *$c \leftrightarrow d$, where $c, d \in O$, called a communication rule,*
  - *$F_i \subseteq O^*$ is a finite set of final states (contents) of agent A_i,*
- *C is a set of labels of colours,*

– *$f \in C$ is the final colour,*
– *B is a set of labels of teams,*
– *B_{colour} is a set of pairs (B_s, c_t) assigning to every team its colour, where $B_s \in B$, $c_t \in C$,*
– *B_{teams} is a set of pairs (A_i, B_s) assigning the label of team $B_s \in B$ to each agent A_i.*

Accepting mode of infinite computations Due to the results of the computational power of APCol systems, it can easily be seen that for finite computations colours and teams do not add more, they can only be used for defining restricted classes of APCol systems. However, this is not the case if we consider unbounded computations.

We say that a string is accepted by an APCol system with coloured teams, if starting with the string as initial string the computation is unbounded and its teams with the final colour are active in an infinite number of steps and the teams of the other colours are active only in a finite number of steps.

Now we provide an illustrative example of APCol system with coloured teams.

Example 1. We construct an APCol system with three teams assigned to three different colours - red, green and orange, simulating work of streets lights connected with speed radar. The green light is on the traffic light at the beginning. If the vehicle is approaching faster than allowed, the traffic light changes to orange and red. In the case that the vehicle stops before the traffic lights (or it drives away while the red is on), the traffic light lights up orange and then green again.The input string for computation is a sequence of signals coming from speed radar. The signals are encoded into symbols in such a way that F means fast speed over limit, S means slow speed within the limits, Z means that car stopped and finally E which means that street is empty. The signals are encoded and inserted into the string with given frequency. Every input string starts with special symbol \$.

The constructed APCol system

$$\begin{aligned}\Pi = (&\{e, E, Z, S, F, o, r, g, \$, R, P\}, e,\\ &A_1, A_2, A_3, \{green, orange, red\}, green,\\ &\{B_1, B_2, B_3\}, \{(B_1, green), (B_2, orange), (B_3, red)\},\\ &\{(A_1, B_1), (A_2, B_2), (A_3, B_3)\})\end{aligned}$$

has three teams - one green, one orange and one red. Each team is formed from only one agent. Agent A_1 has initial configuration ge and the following programs:

$1 : \langle g \leftrightarrow \$; e \leftrightarrow X\rangle \quad X \in \{E, Z, S\}$

$2 : \langle \$ \leftrightarrow g; X \rightarrow e\rangle$

The green team is active only if the current symbol is in accordance with the speed limit or the street is empty. When the speed of the arriving vehicle is over the speed limit, only the orange team can work.

The initial configuration of the agent A_2 is oe and it executes following programs:

$3 : \langle o \leftrightarrow \$; e \leftrightarrow F\rangle$
$4 : \langle \$ \rightarrow \$; F \rightarrow R\rangle$
$5 : \langle \$ \rightarrow \$; R \leftrightarrow o\rangle$

The agent from the orange team consumes symbol F and replaces symbol \$ by R. When this symbol appears in the string, only the red team can work.

The initial configuration of the agent A_3 is re and it performs the following programs:

$6 : \langle r \leftrightarrow R; e \leftrightarrow X\rangle\,,\quad X \in \{E, F, S, Z\}$
$7 : \langle R \rightarrow R; Y \rightarrow e\rangle\,,\ \ Y \in \{F, S\}$
$8 : \langle R \leftrightarrow r; e \rightarrow e\rangle$
$9 : \langle R \rightarrow P; K \rightarrow e\rangle\,,\ \ K \in \{Z, E\}$
$10 : \langle P \leftrightarrow r; e \rightarrow e\rangle$

The agent from the red team consumes symbol R and the neighbouring symbol from the string. The following behaviour of the agent depends on consumed symbol. If the symbol is F or it is S, then the agent puts to the string symbol R and in this way it calls itself to work. In the case of symbol Z or E, (the vehicle stopped or the street is empty) then the agent puts the symbol P to the string and the agent from the orange team has an applicable program.

$11 : \langle \$ \rightarrow \$; o \leftrightarrow P\rangle$
$12 : \langle \$ \rightarrow \$; P \rightarrow e\rangle$
$13 : \langle \$ \leftrightarrow o; e \rightarrow e\rangle$

After executing program 13, symbol \$ appears in the string and it can be consumed by agent from the green team or the orange team. Although the computation over a finite string is not unbounded, but one can assume that if there is no output from the speed radar, encoder puts symbols E into the string with a given frequency (it is similar to the endless tape of Turing machine) and the computation can continue with executing programs of the agent from the green team.

3 APCol Systems with Coloured Teams and Red-Green Counter Machines

In this section we study the interconnection between red-green counter machines and APCol systems with coloured teams. First we present a result where the number of agents within every team is minimal, namely, one.

Theorem 1. *For every red-green counter machine*

$$CM = (m, B, B_{red}, B_{green}, l_0, P, T_{in})$$

we can construct an APCol system

$$\Pi = (O, e, A_1, \ldots, A_n, C, B, B_{colours}, B_{teams})$$

with one agent teams simulating the computations of CM.

Proof. Consider a red-green counter machine $CM = (2, B, B_{red}, B_{green}, l_0, P, T_{in})$ accepting language $L(CM)$. To every such counter machine there exists a red-green counter machine $CM' = (2, B \cup \{l'_0\}, B'_{red}, B'_{green}, l'_0, P, T_{in} \cup \{\#\})$ that accepts language $L(CM') = \# \cdot L(CM)$, where $\{\#\} \cap T_{in} = \emptyset$ and the first instruction of CM' to be executed is instruction $l'_0 : (\text{read}(\#), l_0)$. Then it continues the computation in the same way as machine CM. We construct an APCol system $\mathit{\Pi}$ with coloured teams as follows: all labels from the set $B \cup T_{in}$ are objects of the APCol system. The content of register i is represented by the number of copies of objects i occurring in the string. All teams have one agent only. At the beginning of the computation only one team of agents can work - red team of one agent that generates symbols to the beginning of the string.

Team:	B_1	
Colour:	*Red*	
Agent:	$A_1 = (ee, P_1, \emptyset)$	
Programs:	$1 : \langle e \to \#_1; e \to O_1 \rangle$;	$6 : \left\langle \textcircled{R} \to \boxed{R}; R \leftrightarrow e \right\rangle$;
	$2 : \langle \#_1 \leftrightarrow \#; O_1 \leftrightarrow e \rangle$;	$7 : \left\langle \boxed{R} \to \textcircled{G}; e \to G \right\rangle$;
	$3 : \langle \# \to \#_2; e \to \$ \rangle$;	$8 : \left\langle \textcircled{G} \to \boxed{G}; G \leftrightarrow e \right\rangle$;
	$4 : \langle \#_2 \leftrightarrow e; \$ \leftrightarrow O_1 \rangle$;	$9 : \left\langle \boxed{G} \to \textcircled{L}; e \to X_0 \right\rangle$;
	$5 : \langle O_1 \to R; e \to \textcircled{R} \rangle$;	$10 : \left\langle \textcircled{L} \to \boxed{L}; X_0 \leftrightarrow e \right\rangle$;

Symbol X is an element from the set $\{l, r, g\}$ and it is selected as follows: let l_1 be the currently simulated instruction and let l_2 be the label of the next instruction. If l_2 is a read-instruction and the colour of instruction is red (or green) then $X = r$ (or $X = g$). Otherwise $X = l$.

The APCol system starts its computation on string $\#\omega$. Agent A_1 uses programs $1, 2, 3$ and 4 to replace symbol $\#$ by substring $\#_1\#_2\$$. Then it places three symbols (R, G, X_0) into random positions in the string.

Symbols R and G are consumed by two agents from two teams.

Team:	B_2	Team:	B_3
Colour:	*Red*	Colour:	*Green*
Agent:	$A_2 = (ee, P_2, \emptyset)$	Agent:	$A_3 = (ee, P_3, \emptyset)$
Programs:	$11 : \langle e \leftrightarrow R; e \to e \rangle$;	Programs:	$12 : \langle e \leftrightarrow G; e \to e \rangle$;

Let l_1 be a read-instruction $l_1 : (read(a), l_2)$. We construct two similar teams of different colours to execute the first phase of the simulation of the read-instruction. The agent from such a team checks whether the symbol currently read from the input string is a or not. The team of the working agent has the same colour as the previously simulated instruction.

Team:	B_{X_1} for $X \in \{r, g\}$
Colour:	B_{r_1} is *Red*, B_{g_1} is *Green*
Agent:	$A_{x_1} = (ee, P_{x_1}, \emptyset)$
Programs:	$13 : \langle e \leftrightarrow X_1; e \to e \rangle$;
	$14 : \langle X_1 \to L_1'; e \to e \rangle$;
	$15 : \langle L_1' \leftrightarrow \$; e \leftrightarrow y \rangle$; for all $y \in T_{in}$

Team:	B_2 or B_3
Colour:	B_2 is *Red*, B_3 is *Green*
Agent:	A_2 or A_3; $d \in \{R, G\}$
Programs:	$16 : \langle d \leftrightarrow L_1'; e \to M_1 \rangle$;
	$17 : \langle L_1' \to N_1; M_1 \to M_1 \rangle$;
	$18 : \langle M_1 \leftrightarrow d; N_1 \leftrightarrow e \rangle$;

When agent A_{X_1} successfully finishes its work, then agent from a team with the same colour as l_1 inserts symbol l_2 into the string. In the other case, when the read-instruction cannot be performed, agent from red team starts to be active for an unbounded number of steps.

Team:	B_{X_1} for $X \in \{r, g\}$	
Colour:	B_{r_1} is *Red*, B_{g_1} is *Green*	
Agent:	A_{X_1}	
Programs:	$19 : \langle \$ \leftrightarrow M_1; a \to e \rangle$;	$22 : \langle \$ \leftrightarrow M_1; y \leftrightarrow N_1 \rangle$; $y \in T_{in} - \{a\}$;
	$20 : \langle M_1 \to Q_1; e \leftrightarrow N_1 \rangle$;	$23 : \langle M_1 \to W; N_1 \to e \rangle$;
	$21 : \langle Q_1 \leftrightarrow e; N_1 \to E \rangle$;	$24 : \langle W \leftrightarrow e; e \to e \rangle$;

Team:	B_{l_1}
Colour:	*Red* or *Green* (depends on l_1)
Agent:	$A_{l_1} = (ee, P_{l_1}, \emptyset)$
Programs:	$25 : \langle e \leftrightarrow Q_1; e \to X_2 \rangle$;
	$26 : \langle X_1 \leftrightarrow e; Q_1 \to e \rangle$; $X \in \{l, r, g\}$

Team:	B_4
Colour:	*Red*
Agent:	A_4
Programs:	$27 : \langle e \leftrightarrow W; e \to e \rangle$;
	$28 : \langle W \to W; e \to e \rangle$;

For each ADD-instruction $l_1 : (ADD(r), l_2)$, there are two teams of agents of the same colour as the ADD-instruction has.

Team:	B_{l_1}	
Colour:	*Red* or *Green* (depends on the instruction colour)	
Agent:	A_{l_1}	
Programs:	$29 : \langle e \leftrightarrow l_1; a \to e \rangle$;	$32 : \langle \#_r \leftrightarrow M_1; r \leftrightarrow N_1 \rangle$;
	$30 : \langle l_1 \to L_1; e \to e \rangle$;	$33 : \langle M_1 \to X_2; e \to e \rangle$; $X \in \{l, r, g\}$
	$31 : \langle L_1 \leftrightarrow \#_r; e \to r \rangle$;	$34 : \langle X_2 \leftrightarrow e; e \to e \rangle$;

Team:	B_2 or B_3
Colour:	B_2 is *Red*, B_3 is *Green*
Agent:	A_2 or A_3; $d \in \{R, G\}$
Programs:	$35 : \langle d \leftrightarrow L_1; e \to M_1 \rangle$;
	$36 : \langle M_1 \leftrightarrow d; L_1 \to e \rangle$;

The first agent consumes the corresponding symbol of the actually simulated instruction. At the following steps, the agent rewrites the symbol l_1 to L_1 and exchanges this symbol by $\#_r$. In the same time, the agent generates symbol r. Now it is time for the second team to work. The agent from the second team replaces symbol L_1 by R or G - it depends on the colour of the instruction, rewrites it to symbol M_1 and puts the symbol M_1 to the string instead of symbol R or G. When symbol M_1 appears in the string, then the agent B_1 exchanges it by two symbols - $\#_r$ and r.

For SUB-instruction $l_1 : (SUB(r), l_2, l_3)$, there are two teams of the same colour, too. The first team with one agent is for execution of the instruction and the second team is preparing the symbols for further use (symbol L_1 is replaced with M_1).

Team: B_{l_1}
Colour: *Red* or *Green* (depends on the colour of l_1)
Agent: A_{l_1}
Programs: $37 : \langle e \leftrightarrow l_1; e \rightarrow e \rangle$;
$38 : \langle l_1 \rightarrow L_1; e \rightarrow e \rangle$;
$39 : \langle L_1 \leftrightarrow \#_r; e \leftrightarrow r \rangle$;
$40 : \langle L_1 \leftrightarrow \#_r; e \leftrightarrow Z \rangle$;
$Z \in \{\#_{r+1}, \$\}$

Team: B_2 or B_3
Colour: B_2 is *Red*, B_3 is *Green*
Agent: A_2 or A_3; $d \in \{R, G\}$
Programs: $41 : \langle d \leftrightarrow L_1; e \rightarrow M_1 \rangle$;
$42 : \langle M_1 \leftrightarrow d; L_1 \rightarrow e \rangle$;

The idea of simulation of SUB-instruction is that the agent consumes symbol $\#_r$ together with symbol r - if the counter r is not empty -, or with symbol $\#_{r+1}$ (or \$) - if the counter r is empty and it is not the last counter (or it is the last counter). According to its content, the agent generates the label of the next instruction.

Team: B_{l_1}
Colour: *Red* or *Green* (depends on the colour of l_1)
Agent: A_{l_1}
Programs: $43 : \langle \#_r \rightarrow \#_r; r \rightarrow l_2' \rangle$;
$44 : \langle \#_r \leftrightarrow M_1; l_2' \rightarrow l_2'' \rangle$;
$45 : \langle M_1 \rightarrow e; l_2'' \rightarrow l_2''' \rangle$;
$46 : \langle e \leftrightarrow N_1; l_2''' \rightarrow X_2 \rangle$;
$47 : \langle N_1 \rightarrow e; X_2 \leftrightarrow e \rangle$;
$48 : \langle \#_r \leftrightarrow M_1; Z \leftrightarrow N_1 \rangle$;
$33 : \langle M_1 \rightarrow Y_3; N_1 \rightarrow e \rangle$;
$33 : \langle Y_3 \leftrightarrow e; e \rightarrow e \rangle$;
$X, Y \in \{l, r, g\}$;
$Z \in \{\#_{r+1}, \$\}$

We construct the APCol system

$$\Pi = (O, e, A_1, \ldots, A_n, C, B, B_{colours}, B_{teams}) \text{ with:}$$

– $O = T_{in} \cup \{l_i, l_i', l_i'', l_i''', L_i, L_i', M_i, N_i, g_i, r_i, Q_i | l_i \in H\} \cup \{i | 1 \leq i \leq m\} \cup$
$\cup \{e, G, R, \textcircled{R}, \textcircled{G}, \boxed{R}, \boxed{G}, \textcircled{L}, \boxed{L}, W, \$, \#_1, \#2, O_i\}$,
– $n = |H| + 2 \times$ number of read-instructions $+ 4$
– $B = \{B_j\}, 1 \leq j \leq n$
– $C = \{Red, Green\}$
– The sets $B_{colours}, B_{teams}$ and the agents $A_1, \ldots, A_n$ are defined in the previous part of the text.

The computation of the APCol system starts with string $\$w$. The first steps are done by the red team B_1. Teams B_2 and B_3 must go through initialization before they are used the first time during simulation of the first red or green instruction. It can imply only a finite number of mind changes. After initialization of these two agents, the APCol system goes through the same mind changes as the red-green counter machine CM goes through during the corresponding computation. Therefore, if red-green counter machine CM accepts string w, then APCol system Π accepts it too and vice versa.

□

Although the APCol system from proof of Theorem 1. uses the maximally parallel working mode, its work is limited to the use of one team at each step, therefore, to one agent. As a matter of fact, it works sequentially.

Next we provide another simulation of the red-green counter machines with APCol systems with teams and colours.

Theorem 2. *For every red-green counter machine*

$$CM = (m, B, B_{red}, B_{green}, l_0, P, T_{in})$$

we can construct an APCol system

$$\Pi = (O, e, A_1, \ldots, A_n, C, B, B_{colours}, B_{teams})$$

with at least one team formed from two agents simulating the computations of CM with the same result.

Proof. As in proof of Theorem 1, let us consider red-green counter machine

$$CM = (2, B, B_{red}, B_{green}, l_0, P, T_{in})$$

accepting language $L(CM)$. To every such a red-green counter machine there exists a red-green counter machine

$$CM' = (2, B \cup \{l'_0\}, B'_{red}, B'_{green}, l'_0, P, T_{in} \cup \{\#\})$$

that accepts language $L(CM') = \# \cdot L(CM)$, where $\{\#\} \cap T_{in} = \emptyset$ and the first instruction of the machine CM' to be executed is instruction $l'_0 : (\text{read}(\#), l_0)$. Then, it continues the computation in the same way as machine CM. We construct an APCol system Π with coloured teams as follows: All labels from the set $B \cup T_{in}$ are objects of the APCol system. The content of register i is represented by the number of copies of objects i in the string. At the beginning of the computation only one team of agents can work - red team of one agent that generates symbols to the beginning of the string.

Team:	B_1	
Colour:	*Red*	
Agent:	$A_1 = (ee, P_1, \emptyset)$	
Programs:	$1 : \langle e \to \#_1; e \to O_1 \rangle$;	$4 : \langle \#_2 \leftrightarrow e; \$ \leftrightarrow O_1 \rangle$;
	$2 : \langle \#_1 \leftrightarrow \#; O_1 \leftrightarrow e \rangle$;	$5 : \langle O_1 \to l_0; e \to T \rangle$;
	$3 : \langle \# \to \#_2; e \to \$ \rangle$;	$6 : \langle T \to T; l_0 \leftrightarrow e \rangle$;

The APCol system starts its computation on string $\#\omega$. Agent A_1 uses programs $1, 2, 3$ and 4 to replace symbol $\#$ by substring $\#_1\#_2\$$. Then it places symbol l_0 into some random position in the string.

Let l_1 be a read-instruction $l_1 : (read(a), l_2)$. We construct a team of the same colour as the read-instruction has. The team is formed from two agents. Because they work as a team, either they both execute their programs or none of them works.

Team:	B_{l_1}		
Colour:	*Red* or *Green* (it depends on the colour of l_1)		
Agent:	$A_{a_1} = (ee, P_{a_1}, \emptyset)$	Agent:	$A_{b_1} = (ee, P_{b_1}, \emptyset)$
Programs:	$7 : \langle e \leftrightarrow l_1; e \to e\rangle$;	Programs:	$11 : \langle e \to R_1; e \to e\rangle$;
	$8 : \langle l_1 \to \$; e \to e\rangle$;		$12 : \langle R_1 \leftrightarrow \$; e \to x\rangle$;
	$9 : \langle \$ \leftrightarrow R_1; e \to e\rangle$;		for all $x \in T_{in}$
	$10 : \langle R_1 \to e; e \to e\rangle$;		$13 : \langle \$ \to l_2; a \to e\rangle$;
			$14 : \langle \$ \to W; y \to e\rangle$;
			for all $y \in T_{in} - \{a\}$
			$15 : \langle l_2 \leftrightarrow e; e \to e\rangle$;
			$16 : \langle W \leftrightarrow e; e \to e\rangle$;

Although agent A_{b_1} has an applicable program it must stay inactive until the first agent has an applicable program, too, i.e., until symbol l_1 appears in the string.

Team:	B_1
Colour:	*Red*
Agent:	A_1
Programs:	$17 : \langle e \leftrightarrow W; T \to T\rangle$;
	$18 : \langle W \to W; T \to T\rangle$;

When the read-instruction cannot be performed, agent A_1 from red team starts working for an unbounded number of steps.

For each ADD-instruction $l_1 : (ADD(r), l_2)$, there is one team of agents of the same colour as the ADD-instruction has.

Team:	B_{l_1}		
Colour:	*Red* or *Green* (it depends on the colour of l_1)		
Agent:	$A_{a_1} = (ee, P_{a_1}, \emptyset)$	Agent:	$A_{b_1} = (ee, P_{b_1}, \emptyset)$
Programs:	$19 : \langle e \leftrightarrow l_1; e \to L_1\rangle$;	Programs:	$24 : \langle e \to r'; e \to e\rangle$;
	$20 : \langle L_1 \leftrightarrow \#_r; l_1 \to K_1\rangle$;		$25 : \langle r' \to r; e \to M_1\rangle$;
	$21 : \langle \#_r \to \#_r; K_1 \to K_2\rangle$;		$26 : \langle M_1 \leftrightarrow L_1; r \leftrightarrow e\rangle$;
	$22 : \langle \#_r \leftrightarrow M_1; K_2 \to K_3\rangle$;		$27 : \langle L_1 \to l_2; e \to e\rangle$;
	$23 : \langle M_1 \to e; K_3 \to e\rangle$;		$28 : \langle l_2 \leftrightarrow e; e \to e\rangle$;

The first agent consumes the symbol corresponding to the actually simulated instruction. In the same time, the second agent starts to generate symbol r. At the following steps, the first agent rewrites symbol l_1 to L_1 and exchanges this symbol by $\#_r$. The first agent can put symbol $\#_r$ back to the string only by replacing it by symbol M_1 generated by the second agent. The second agent inserts the label of the next instruction at some random place in the string.

For SUB-instruction $l_1 : (SUB(r), l_2, l_3)$, there is one team of the same colour as the instruction has.

Team:	B_{l_1}		
Colour:	*Red* or *Green* (it depends on the colour of l_1)		
Agent:	$A_{a_1} = (ee, P_{a_1}, \emptyset)$	Agent:	$A_{b_1} = (ee, P_{b_1}, \emptyset)$
Programs:	$29 : \langle e \leftrightarrow l_1; e \to L_1 \rangle$;	Programs:	$37 : \langle e \to M_1; e \to K_1 \rangle$;
	$30 : \langle L_1 \leftrightarrow \#_r; l_1 \to K_1 \rangle$;		$38 : \langle M_1 \to M_1; K_1 \to e \rangle$;
	$31 : \langle \#_r \to \#_r; K_1 \to K_2 \rangle$;		$39 : \langle M_1 \leftrightarrow L_1; e \leftrightarrow d \rangle$;
	$32 : \langle \#_r \leftrightarrow M_1; K_2 \to K_2 \rangle$;		for all $d \in \{r, \#_{r+1}, \$\}$
	$33 : \langle M_1 \to M_1'; K_2 \to K_2 \rangle$;		$40 : \langle L_1 \to N_1; r \to K_1 \rangle$;
	$34 : \langle M_1' \leftrightarrow N_1; K_2 \to K_2 \rangle$;		$41 : \langle L_1 \to N_1; d' \to d' \rangle$;
	$35 : \langle N_1 \to e; K_2 \to K_2 \rangle$;		for all $d' \in \{\#_{r+1}, \$\}$
	$36 : \langle e \to e; K_2 \to e \rangle$;		$42 : \langle N_1 \leftrightarrow \#_r; K_1 \to K_2 \rangle$;
			$43 : \langle N_1 \leftrightarrow \#_r; d' \to d' \rangle$;
			$44 : \langle \#_r \to \#_r; K_1 \to K_2 \rangle$;
			$45 : \langle \#_r \to \#_r; K_2 \to K_3 \rangle$;
			$46 : \langle \#_r \leftrightarrow M_1'; K_3 \to l_2 \rangle$;
			$47 : \langle \#_r \to l_2; e \to e \rangle$;
			$48 : \langle N_1 \leftrightarrow \#_r; d' \leftrightarrow e \rangle$;
			$49 : \langle \#_r \to \#_r; e \to K \rangle$;
			$50 : \langle \#_r \leftrightarrow M_1'; K \to l_3 \rangle$;
			$51 : \langle l_2 \leftrightarrow e; M_1' \to e \rangle$;
			$52 : \langle l_3 \leftrightarrow e; M_1' \to e \rangle$;

The idea of simulation of SUB-instruction is that agent consumes symbol $\#_r$ together with the right neighbouring symbol. According to content of the agent, it generates the label of the next instruction.

We construct the APCol system

$$\Pi = (O, e, A_1, \ldots, A_n, C, Green, B, B_{colours}, B_{teams}) \text{ with:}$$

– $O = T_{in} \cup \{l_i, L_i, M_i, M_1', N_i, R_i | l_i \in H\} \cup \{i, i' | 1 \leq i \leq m\} \cup$

$\cup \{e, K_1, K_2, K_3, W, \$, \#_1, \#2\}$,

– $n = 2 \times |H| + 1$

– $B = \{B_j\}, 1 \leq j \leq p;\ p = |H| + 1$

– $C = \{Red, Green\}$

– The sets $B_{colours}, B_{teams}$ and the agents $A_1, \ldots, A_n$ are defined in the previous part of the text.

The computation of the APCol system starts with string $\$w$. The first steps are done by the red team B_1. After initialization, the APCol system goes through the same mind changes as the counter machine goes through during the corresponding computation. Therefore, if red-green counter machine CM accepts string w, then APCol system Π accepts string $\#w$ too, and vice versa.

$\square$

4 Conclusions

In this paper, we investigated the possibility of "going beyond" Turing in the terms of APCol systems. We introduced the notion of teams of agents as a restriction for the maximal parallelism of computation. In addition, we assigned a colour to each team. The unbounded computation was described by the sequence of the colours associated to the acting teams. We have shown that we can simulate red-green counter machines with APCol systems with two-coloured teams. Red-green counter machines are computing devices with infinite run on finite input that exceed the power of Turing machines.

As we mentioned in the Introduction, there are concepts in P systems theory which are motivated and mimic the behaviour of red-green Turing machines, for example [2]. The proofs of the theorems in Section 3 demonstrate that finite communities of very simple and very small computing devices (i.e. agents and programs) in a suitable environment and using a simple cooperation protocol (based on colours) can produce a behaviour which may not be computable in the sense of Turing machines. These results add further information on the behaviour of communities of agents and ideas to constructs networks of computing agents.

Acknowledgments. This 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, by SGS/13/2016 and by Grant No. 120558 of the National Research, Development, and Innovation Office, Hungary.

References

1. Alhazov, A., Aman, B., Freund, R., Păun, G.: Matter and Anti-matter in Membrane Systems. In: Jürgensen, H., Karhumäki, J., Okhotin, A. (eds.) Descriptional Complexity of Formal Systems: 16th International Workshop, DCFS 2014, Turku, Finland, August 5-8, 2014. Proceedings. pp. 65–76. Springer International Publishing, Cham (2014),
2. Aman, B., Csuhaj-Varjú, E., Freund, R.: Red-Green P Automata. In: Gheorghe, M. et al. (eds.): CMC 2014, LNCS, vol. 8961, pp. 139–157, Springer (2014)
3. Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E.: A Class of Restricted P Colonies with String Environment. Natural Computing 15(4), 541–549 (2016),
4. Cienciala, L., Ciencialová, L.: P Colonies and Their Extensions. In: Kelemen, J., Kelemenová, A. (eds.) Computation, Cooperation, and Life – Essays Dedicated to Gheorghe Paun on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 6610, pp. 158–169. Springer-Verlag, Berlin Heidelberg (2011),

5. Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E.: Towards on P Colonies Processing Strings. In: Proc. BWMC 2014, Sevilla, 2014. pp. 102–118. Fénix Editora, Sevilla, Spain (2014)
6. Csuhaj-Varjú, E.: Extensions of P Colonies (Extended Abstract). In: Leporati, A. and Zandron, C. (Eds.) Proc. CMC17, Milan, 2014. pp. 281–286. University Milano-Bicocca & IMCS, Italy (2014)
7. 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)
8. 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)
9. 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)
10. Kelemen, J., Kelemenová, A.: A Grammar-Theoretic Treatment of Multiagent Systems. Cybern. Syst. 23(6), 621–633 (Nov 1992),
11. Minsky, M. L.: Computation: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, NJ, 1967.
12. van Leeuwen, J., Wiedermann, J.: Computation as an Unbounded Process. Theoretical Computer Science 429, 202 – 212 (2012),
13. Meduna, A., Zemek, P.: Jumping Finite Automata. Int. J. Found. Comput. Sci. 23(7), 1555–1578 (2012)
14. Păun, Gh., Rozenberg, G., Salomaa, A.(eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., New York, NY, USA (2010)
15. Rozenberg, G., Salomaa, A.(eds.): Handbook of Formal Languages I-III. Springer Verlag., Berin-Heidelberg-New York (1997)

On Languages Generated by Spiking Neural P Systems with Structural Plasticity

Ren Tristan A. de la Cruz[1], Francis George C. Cabarle[1,2] and Xiangxiang Zeng[2]

[1]Algorithms and Complexity Laboratory
Department of Computer Science
University of the Philippines
Diliman, Quezon City, Philippines;
[2]School of Information Science and Technology
Xiamen University
Xiamen 361005, Fujian, China.
rentristandelacruz@gmail.com, fccabarle@up.edu.ph, xzeng@xmu.edu.cn

Abstract. Spiking neural P systems are a model of computation inspired by the working of the brain. Its main components are *neurons* that can be connected to one another to form a system. Neurons redcommunicate by sending each other *spikes*. After the introduction of the original model in 2006, different variants of SNP system model have been developed with each variant modifying and/or adding components to the model. The original SNP system and its variants have been used as language generators. In this work, we use a variant of SNP system known as SNP system *with structural plasticity* (SNPSP system) for language generation. The SNPSP system model adds the ability to dynamically create and delete connections (synapses) between neurons. We devise construction procedures for creating SNPSP systems that generate finite (FIN), regular (REG), and recursively enumerable (RE) languages.

Keywords: Language Generation, Spiking Neural P Systems, Membrane Computing, Structural Plasticity

1 Introduction

A family of related models of computation known as *P systems* is the topic of the area of theoretical computer science known as *membrane computing*. Membrane computing studies biologically-inspired unconventional computing models [12]. P systems include models of computation inspired by the mechanism inside of a biological cell. They also include models inspired by the workings of a group of connected cells (tissue-like models) and models inspired by the workings of a group of neurons (neural-like models).

Spiking neural P systems (SNP systems) are neural-like P systems introduced in 2006 [5]. The SNP system model is inspired by the workings of the brain. It is an unconventional model of computation whose main computing elements are called *neurons*. The neurons also serve as storage, each neuron storing a single number called a *spike count*. An SNP system is a collection of such neurons that can be connected to each other using synapses. This model will later be called *classic* SNP system.

The classic SNP system is a computing model that is based on an extremely simplified and highly abstracted mechanism of the brain. There are other components and mechanisms of the brain that can be used as inspirations for additional components/mechanisms of the SNP system model. After being introduced in 2006, many other variants of SNP systems have been developed.

The classic SNP system uses only a single signal known as *spike* for communicating information from one neuron to other neurons. In [4], the idea of using different level of signals (sending multiple spikes) is added to the classic SNP system. The resulting model is known as SNP system *with extended rules*.

The idea of *neurogenesis*, where new neurons can be created by the system, is introduced in [10]. The model is called SNP system *with neuron division and budding*. In [11], the glial cells known as *astrocytes* are added to the classic SNP system. In this variant, there exist entities known as astrocytes that are connected to the synapses (connections between neurons). An astrocyte 'regulates' the spikes going through the set of synapses where it is connected. The model is called SNP system *with astrocytes*.

The idea of 'anti-spike' is added in [9]. The idea is mainly inspired by the concepts of anti-matter and matter-antimatter annihilation. A new type of signal called 'anti-spike' is introduced to the SNP system model. Anti-spike signals 'annihilates' spike signals and there is an implicit rule that when anti-spike and a spike are in the same neuron they cancel out each other. The resulting model is known as SNP system *with anti-spikes*.

A more recently introduced variant is known as SNP system with *structural plasticity* (SNPSP system) [2]. The main inspiration for this variant is the idea of *synaptogenesis*. Synaptogenesis is the idea of the brain being *structurally plastic*, having the ability to rewire itself by 'creating' new synapses or 'deleting' existing synapses between neurons. The classic SNP system and its variants have been used as language generating models. The language generation using classic SNP systems has been studied in [3]. Some literature on language generation of SNP variants are: [4] for SNP systems with extended rules, [6] for SNP systems with astrocytes, [7] for SNP systems with anti-spikes.

The main purpose of this paper is to study the language generation using the SNPSP system model. Specifically, we wanted to see how SNPSP systems can be used to generate finite languages (FIN), regular languages (REG), and recursively enumerable languages (RE).

2 Prerequisites

In this Section we discuss preliminary concepts needed to understand the discussion of results in Section 3. Section 2.1 discusses languages, operations between languages and regular expressions. Section 2.2 discusses register machines , and section 2.3 discusses spiking neural P systems with structural plasticity, the main computing model being studied.

2.1 Languages and Regular Expressions

An alphabet V is a set of symbols. A string s is a concatenation of symbols. If a string s uses only symbols of a particular alphabet V, that string is said to be *over the alphabet* V. If s is a string over V, the notation $|s|$ is used to denote the length of s while $|s|_a$ where $a \in V$ is used to denote the number of occurrences of symbol a in s.

A language L is a set of strings. When talking about languages , the term *word* can be used as a synonym for string. If V is a finite alphabet, then V^* is a language that contains all strings over the alphabet V. V^* also includes the empty string, denoted as λ, which is the string with no symbols. V^+ is a language that contains all non-empty strings over V.

Since languages are sets , the usual set operations (union, intersection, subtraction) can be applied to languages to produce a new language , e.g. $L_3 = L_1 \cup L_2$, $L_3 = L_1 \cap L_2$. Concatenation of languages L_1 and L_2, denoted as L_1L_2, will produce the language $L = \{s_1s_2|s_1 \in L_1, s_2 \in L_2\}$. The unary operator * applied to some language L will produce a new language L^* that contains the empty string λ and all words created by concatenating words of L.

If V_1 and V_2 are alphabets, a morphism $h : {V_1}^* \to V_2$ is a mapping that satisfies the condition $h(uv) = h(u)h(v)$ for $u, v \in {V_1}^*$. For a morphism $h : {V_1}^* \to V_2$ and a string $y \in V_2$, $h^{-1}(y)$ is defined as $h^{-1}(y) = \{x \in {V_1}^*|h(x) = y\}$; this mapping from ${V_2}^*$ to the power set of ${V_1}^*$ is extended in the natural way to languages over V_2 and is called the inverse morphism associated with h. A morphism $h : V_1 \to V_1$ is called projection if $h(a) \in \{a, \lambda\}$ for each $a \in V_1$.

A regular expression is a string that can represent a specific language over some alphabet V. Any language that can be defined by a regular expression is called a *regular language*. The set of all regular languages is usually denoted as *REG*.

Given a regular expression E , the language it defines is denoted as $L(E)$. If a regular expression E defines a language over V, then E itself, is a string over the alphabet $V \cup \{(,), *, +\}$. Given an alphabet V, the symbol λ, any symbol $a \in V$, and the symbol $\emptyset$are regular expressions. λ defines the language $\{\lambda\}$, $\emptyset$ defines the empty language $\{\}$, and a defines the language $\{a\}$. If E_1 and E_2 are regular expressions, then (E_1), E_2^*, E_1E_2, $E_1 + E_2$ are also regular expressions. (E_1) defines the language $L(E_1)$, E_2^* defines the language $L(E_2)^*$, E_1E_2 defines that language $L(E_1)L(E_2)$, and $E_1 + E_2$ defines the language $L(E_1) \cup L(E_2)$.

2.2 Register Machines

A *register machine* is a simple model of computation that uses *registers* to store information. A register stores a single non-negative integer. The register machine performs two primary types of instruction: incrementation and decrementation of values in a register. A third type of instruction known as $HALT$ will stop the entire operation of the machine when executed.

Formally, a register machine M is a construct $M = (m, I, l_0, l_h, R)$, where

- m is the number of registers.
- I is the set of instruction labels.
- l_o is the label of the initial instruction.
- l_h is the label of the halt instruction.
- R is the set of all instructions.

Each label from I labels only one instruction from R.

The instructions have the following forms:

- $l_i : (ADD(r), l_j, l_k)$. When this instruction is executed, the machine will increment the number stored in register r by 1 then it will non-deterministically select between the instruction labeled as l_j or the instruction labeled as l_k to execute next.
- $l_i : (SUB(r), l_j, l_k)$. When this instruction is executed, the machine will decrement the number stored in register r by 1 if the value is not zero and instruction labeled as l_j will be the next instruction to execute. If the value in r is zero, the machine will not perform decrementation , it will simply select the instruction labeled as l_k as the next instruction.
- $l_h : HALT$. This is the halt instruction. It will stop the operations of the machine once executed.

The machine will start with all its registers empty (all the registers store zero) and executes the instruction labeled as l_0. The machine will continue to execute instructions until it executes the halt instruction labeled as l_h. If the machine halts, then the value in the first register is said to be generated by the machine M. The set of numbers that register machine M can generate is denoted by $N(M)$. It has been proven in [8] that register machines compute all set of numbers that are computable using Turing machines (denoted as NRE).

2.3 Spiking Neural P Systems with Structural Plasticity

Spiking neural P systems with structural plasticity (SNPSP systems) were introduced in [2]. They are variants of classic spiking neural P systems (SNP systems) from [5].

Formally, an SNPSP system Π of degree $m \geq 1$ is a construct $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, in, out)$, where

- $O = \{a\}$ is a singleton alphabet containing only the symbol a. a is called a *spike*.
- $\sigma_1, \ldots, \sigma_m$ are the *neurons* of the system. A neuron σ_i $(1 \leq i \leq m)$ has the form (n_i, R_i). n_i is a non-negative integer that indicates the initial number of spikes in σ_i. n_i is represented by the string a^{n_i} over the alphabet O. R_i is a finite set of rules with the following forms:

1. **Spiking Rule:** $E/a^c \to a$ where E is a regular expression over O and $c \geq 1$. When $E = a^c$, the rule can be written as $a^c \to a$.
2. **Plasticity Rule:** $E/a^c \to \alpha k(i, N)$ where $c \geq 1$, $\alpha \in \{+, -, \pm, \mp\}$, $N \subseteq \{1, \ldots, m\}$, and $1 \leq k \leq |N|$. When $E = a^c$, the rule can be written as $a^c \to \alpha k(i, N)$.

– $syn \subseteq \{1, \ldots, m\} \times \{1, \ldots, m\}$, with $(i, i) \notin syn$, is the set of initial *synapses* between neurons.
– in, out are neuron labels that indicate the input and output neurons , respectively.

The semantics of SNPSP systems is as follows. For every time step, each neuron of an SNPSP system Π will check if any of its rules is applicable. Activation requirements of a rule are specified as E/a^c. A rule is applicable if the following conditions are met: (1) the number of spikes in the neuron (that contains the rule), represented by a^n, falls under that pattern of the regular expression E (symbolically $a^n \in L(E)$) and (2) the number of spikes in the neuron is at least c.

It is possible that multiple rules are applicable in a neuron at a given time. This occurs when the languages defined by the regular expressions of the rules intersect. i.e. Rule 1: $E_1/a^{c_1} \to a$, rule 2: $E_2/a^{c_2} \to a$ and $L(E_1) \cap L(E_2) \neq \varnothing$. When multiple rules are applicable in a neuron, the neuron will non-deterministically select a rule to activate. When a rule is activated c spikes are consumed in the neuron.

When a spiking rule is activated at some neuron σ_i, all neurons σ_j such that $(i, j) \in syn$ will receive a spike from σ_i.

When a plasticity rule $E/a^c \to \alpha k(i, N)$ is activated in the neuron, the neuron will perform one of the following actions:

1. $(\alpha = +)$ Add a set of k synapses from σ_i to some k neurons whose labels are specified in N
2. $(\alpha = -)$ Delete a set of k synapses that connect σ_i to some neurons whose labels are specified in N
3. $(\alpha = \pm)$ At time step t, add a set of k synapses from σ_i to some k neurons whose labels are specified in N, then in the next time step $t+1$, delete those same k synapses
4. $(\alpha = \mp)$ At time step t, delete a set of k synapses that connect σ_i to some neurons whose labels are specified in N, then in the next time step $t+1$ add those same k synapses back

Let $P(i) = \{j | (i, j) \in syn\}$. It is the set of neuron labels (j) such that σ_i is connected to σ_j. If a plasticity rule is activated and is specified to add k synapses there will be cases when it can only add less than k synapses. This occurs when most of the neurons specified in N already have connections from σ_i , i.e. $|N - P(i)| < k$. The rule will connect σ_i to the remaining neurons specified in N that are not in $P(i)$. If $|N - P(i)| = 0$ then there are no more synapses to add. If $|N - P(i)| = k$ then there are exactly k synapses to add.

When $|N - P(i)| > k$ then the rule will non-deterministically select k neurons from $N - P(i)$ and connect σ_i to those neurons.

Additionally we note the following: When a synapse is created at time step t, connecting σ_i to σ_j, a spike is sent to σ_j at the same time step t.

Similar cases can occcur when deleting synapses. If $|P(i)| < k$, then a rule will only delete less than k synapses that connect σ_i to neurons specified in N that are also in $P(i)$. If $|P(i)| = 0$, then there will be no synapses to delete. If $|P(i) \cap N| = k$ then the rule will delete exactly k synapses that connect σ_i to neurons specified in N.

When $|P(i) \cap N| > k$ then the rule will non-deterministically select k synapses that connect σ_i to neurons in N and delete those synapses.

A plasticity rule with $\alpha \in \{\pm, \mp\}$ will be active for two time steps. When such rule is activated at time step t it will be active until time step $t+1$. During time steps t and $t+1$ no other rules can be activated but the neuron can still receive spikes.

For language generation, one neuron is designated as the output neuron. The output neuron is said to have a synapse to the *environment*. When a spiking rule is activated at time step t by the output neuron, the system is said to generate the symbol '1'. When no spiking rules are activated in the output neuron, the system is said to generate the symbol '0'. The string generated by the system is the concatenation of the symbols generated until the entire system halts. The system halts when there are no active rules and no other rules in any neuron can be activated.

When considering only finite strings, which are the ones considered in this paper, if the system does not halt on a given run then no string is generated for that run. Using this interpretation, only binary languages are directly generated by the system.

3 Results

In this section we present four main results for using SNPSP systems to generate the following languages: (1) finite binary languages of the form $L' = \{0\}L$ where L is a finite binary language, (2) finite binary languages, (3) regular binary languages, and (4) recursively enumerable binary languages.

Since SNPSP systems are used to generate those languages, only binary languages that do not contain λ are considered. λ is not considered since the output neuron of an SNPSP system only generates the symbols '0' (no spike is sent to the environment) or '1' (a spike is sent to the environment).

3.1 SNPSP Systems for Finite Languages of the Form {0}L

Let B be the binary alphabet $\{0, 1\}$. This notation will be used in the other sections as well.

Theorem 1. *If $L \in FIN$ and $L \subseteq B^+$, then there is an SNPSP system Π that generates words in L' where $L' = \{0\}L$.*

Proof. Let $L = \{b_1, b_2, b_3, \ldots, b_n\}$ and $L' = \{b'_i | b'_i = 0b_i, b_i \in L\}$. Let $l_{max} = \max\{|b_i|\}$ (for $1 \leq i \leq n$) be the length of the longest binary string in L. The SNPSP system Π for generating L' is shown in Figure 1.

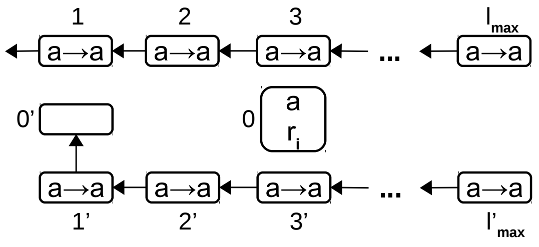


Fig. 1. Neurons of Π (FIN Language Generator)

There will be an initial $(2 \cdot l_{max})$ synapses in Π. For $i \in \{1, \ldots, l_{max} - 1\}$, a synapse from σ_{i+1} to σ_i will be created. For $i' \in \{0', 1', \ldots, (l_{max} - 1)'\}$, a synapse from $\sigma_{(i+1)'}$ to $\sigma_{i'}$ will be created.

Every neuron, except σ_0 and $\sigma_{0'}$, will contain only one rule: "$a \rightarrow a$". $\sigma_{0'}$ will have no rules. σ_0 will have n plasticity rules, one rule r_i for each word $b_i \in L$. The rules are formulated in the following way:

– For each b_i, there will be a rule r_i in σ_0.
– Form of the rule: $r_i : a/a \rightarrow +k_i(0, N_i)$.
– $k_i = (|b_i|_1 + 1)$ is the number of '1' in the string b_i plus 1.
– $N_i = O_i \cup \{|b_i|'\}$ where $O_i = \{x|$ if the x^{th} symbol of b_i is '1'$\}$. O_i is a set of neuron labels from the "neuron chain" ($\sigma_1 \Leftarrow \ldots \Leftarrow \sigma_{l_{max}}$). It will represent the positions of '1' in b_i. $|b_i|'$ is a neuron label, from the neuron chain ($\sigma_{0'} \Leftarrow \ldots \Leftarrow \sigma_{l'_{max}}$), that will represent the length of b_i.

σ_0 will have one initial spike. Using this spike, σ_0 can non-deterministically select which rule to activate and therefore which word to generate (all plasticity rules have the same regular expression $E = a$).

When activated, the plasticity rule r_i will connect (i.e. create a synapse from) σ_0 to some of the neurons in the neuron chain ($\sigma_1 \Leftarrow \ldots \Leftarrow \sigma_{l_{max}}$). r_i will send a spike to each neuron in the chain representing the positions of '1' in b_i. If the j^{th} symbol in b_i is '1', then σ_0 will connect to σ_j thus providing a spike to σ_j. After activating r_i, the distribution of spikes in the neuron chain will resemble the binary string b_i.

r_i will also connect σ_0 to a single neuron in the chain ($\sigma_{0'} \Leftarrow \ldots \Leftarrow \sigma_{l'_{max}}$). This neuron represents the length of the string b_i.

At the first time step ($t = 0$), σ_0 activates r_i while the output neuron σ_1 generates '0' since it does not spike. From time step $t = 1$ until time step $|b_i|$, the neuron chain ($\sigma_1 \Leftarrow \ldots \Leftarrow \sigma_{l_{max}}$) will generate the symbols of b_i. The spike sent (at $t = 0$) to $\sigma_{|b_i|'}$ will prevent the SNPSP system Π from halting before the entire string $0b_i$ is generated. This is for cases where b_i has a substring $s \in \{0\}^+$. □

Example. $L = \{b_1 = 000111, b_2 = 10100\}$. $l_{max} = |b_1| = 6$. The SNPSP system Π will have the neuron chains: $(\sigma_1 \Leftarrow \ldots \Leftarrow \sigma_6)$ and $(\sigma_{0'} \Leftarrow \ldots \Leftarrow \sigma_{6'})$. σ_0 will contain two rules, r_1 for generating b_1 and r_2 for generating b_2. r_1 is the plasticity rule $a/a \to +4(0, \{4, 5, 6\} \cup \{6'\})$. r_2 is the plasticity rule $a/a \to +3(0, \{1, 3\} \cup \{5'\})$.

3.2 SNPSP Systems for Finite Languages

In section 3.1, we showed that it is possible to construct an SNPSP system that can generate any finite binary language with words that start with the symbol '0' ($L' = \{0\}L$ where L is some finite non-empty binary language).In this section we will show that SNPSP system can be used to generate any finite non-empty binary language even those that contain words that start with '1'.

Theorem 2. *If $L \in FIN$ and $L \subseteq B^+$, then there is an SNPSP system Π that generates words in L.*

Proof. The SNPSP system Π for generating L will only have two neurons: σ_0 and σ_1. σ_0 will contain all the rules while σ_1 will have no rules. σ_0 is also the designated output neuron. There are no synapses in Π. σ_0 will contain rules with the following forms:

- Form 1: $E_i/a^{c_i} \to a$. This is a spiking rule used for generating the symbol '1'.
- Form 2: $E_i/a^{c_i} \to -1(0, \{1\})$. This plasticity rule will act similar to classic SNP system's *forgetting rules* (which are not use in SNPSP systems), a rule with the form "$a^s \to \lambda$" that consumes s spikes and performs no other operations. During activation, the plasticity rule will consume c_i spikes and will perform no other action since it cannot remove a non-existing synapse connecting σ_0 to σ_1.

Let $L = \{b_1, b_2, b_3, \ldots, b_n\}$ and $l_{max} = \max\{|b_i|\}$ (for $1 \le i \le n$) be the length of the longest binary string in L. For every $b_i \in L$ there will be 3 rules added to σ_0: $r_{0,i}, r_{i,0}, r_{i,1}$.

- $r_{0,i}$ is the *initial rule* associated with the string b_i. It is of form 1 if the initial symbol of b_i is '1', otherwise it is of form 2. From the name itself, initial rule, it will be the first rule activated and it will generate the first symbol of b_i.
- $r_{i,0}$ is the rule for generating the '0' symbols of b_i, excluding the first symbol (if it happens to be '0'). It is of form 2.
- $r_{i,1}$ is the rule for generating the '1' symbols of b_i, excluding the first symbol (if it happens to be '1'). It is of form 1.

Let $P_i = \{p|$ if p^{th} symbol of b_i is '1'$\}$. It is the set of numbers representing the positions of '1' in b_i. Let $Q_i = \{q|$ if q^{th} symbol of b_i is '0'$\}$. It is the set of numbers representing the positions of '0' in b_i. Let $P'_i = P_i - \{1\}$ and $Q'_i = Q_i - \{1\}$. Rules $r_{i,0}$ and $r_{i,1}$ will have the following forms:

– $r_{i,0} : E_{i,0}/a \to -1(0, \{1\})$ where $E_{i,0} = \bigcup_{q_j \in Q'_i} a^{i \cdot l_{max} - (q_j - 2)}$
– $r_{i,1} : E_{i,1}/a \to a$ where $E_{i,1} = \bigcup_{p_j \in P'_i} a^{i \cdot l_{max} - (p_j - 2)}$

Rules $r_{0,i}$ will have the from:

– $r_{0,i} : a^{(n+1) \cdot l_{max}} / a^{(n+1-i) \cdot l_{max}} \to -1(0, \{1\})$, if the first symbol of b_i is '0'.
– $r_{0,i} : a^{(n+1) \cdot l_{max}} / a^{(n+1-i) \cdot l_{max}} \to a$, if the first symbol of b_i is '1'.

σ_0 will initially have $(n+1) \cdot l_{max}$ number of spikes. Since the initial rules $r_{i,0}$ have the same regular expression $E = a^{(n+1) \cdot l_{max}}$, σ_0 will non-deterministically select which of the initial rules it will activate. This selection determines the first symbol of b_i generated.

$r_{0,i}$ will consume $((n+1-i) \cdot l_{max})$ spikes and will generate the first symbol of b_i. $(i \cdot l_{max})$ spikes will remain after activating $r_{0,i}$. Rules $r_{i,0}$ and $r_{i,1}$ can only activate when the number of spikes is from $(i \cdot l_{max})$ to $((i-1) \cdot l_{max} + 1)$. Specifically, $r_{i,0}$ and $r_{i,1}$ will only activate when the spike count is from $(i \cdot l_{max})$ to $(i \cdot l_{max} - (|b_i| - 2))$.

After $r_{0,i}$ generates the first symbol of b_i, the rest of the string will be generated by $r_{i,0}$ and $r_{i,1}$.

□

Example. $L = \{b_1 = 101, b_2 = 01001, b_3 = 1100\}$

– $l_{max} = |b_2| = 5$.
– Initial spike count: $20 = (3+1) \cdot l_{max}$
– $P_1 = \{1,3\}$, $Q_1 = \{2\}$ and $P'_1 = \{3\}$, $Q'_1 = \{2\}$.
– $P_2 = \{2,5\}$, $Q_2 = \{1,3,4\}$ and $P'_2 = \{2,5\}$, $Q'_2 = \{3,4\}$.
– $P_3 = \{1,2\}$, $Q_3 = \{3,4\}$ and $P'_3 = \{2\}$, $Q'_3 = \{3,4\}$.
– Initial rules:
  $r_{0,1} : a^{20}/a^{15} \to a$
  $r_{0,2} : a^{20}/a^{10} \to -1(0, \{1\})$
  $r_{0,3} : a^{20}/a^{5} \to a$
– '0' generating rules $(r_{i,0})$:
  $r_{1,0} : (a^5)/a \to -1(0, \{1\})$
  $r_{2,0} : (a^9 + a^8)/a \to -1(0, \{1\})$
  $r_{3,0} : (a^{14} + a^{13})/a \to -1(0, \{1\})$
– '1' generating rules $(r_{i,1})$:
  $r_{1,1} : (a^4)/a \to a$
  $r_{2,1} : (a^{10} + a^7)/a \to a$
  $r_{3,1} : (a^{15})/a \to a$

3.3 SNPSP Systems for Regular Languages

Theorem 3. *If $L \in B^+$ and $L \in REG$, then there is an SNPSP system Π and morphism $h_1 : B^* \to B^*$ such that $L = h_1^{-1}(L(\Pi))$.*

Proof. Let $G = (N, B, S, P)$ be a *right-linear* grammar over the binary alphabet B.

- $N = \{N_1, N_2, \ldots, N_m\}$ is the set of non-terminal symbols. $|N| = m$.
- $B = \{0, 1\}$ is the alphabet or set of terminal symbols.
- $S \in N$ is the axiom or start (non-terminal) symbol. Let $S = N_m$.
- P is the set of k right-linear production rules having the following forms:
  1. $R_i : N_{p_i} \to bN_{q_i}$, $b \in B$, $1 \le p_i, q_i \le m$, $1 \le i \le k$
  2. $R_i : N_{p_i} \to b$, $b \in B$, $1 \le p_i \le m$, $1 \le i \le k$

The SNPSP system Π will simulate the word generation process of grammar G. Π has $(m + 4)$ neurons. Figure 2 below shows all the neurons of Π and the initial set of synapses between its neurons.

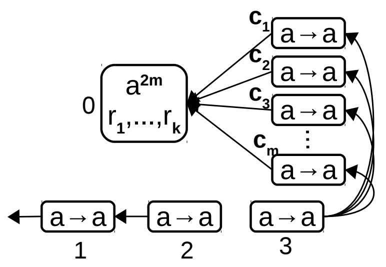


Fig. 2. Neurons of Π for generating languages in REG

Every neuron in Π, except σ_0, has zero initial spikes and contains the spiking rule "$a \to a$". σ_0 has $2m$ spikes and contains k plasticity rules. Each plasticity rule r_i in σ_0 corresponds to a production rule R_i in grammar G.

For a non-terminal production rule $R_i : N_{p_i} \to bN_{q_i}$ the corresponding plasticity rule r_i is added to σ_0. Plasticity rule r_i will have the form:

- If $b = 1$, $r_i : a^{m+p_i}/a^{m+p_i-q_i} \to \pm 3(0, \{1, 2, 3\})$.
- If $b = 0$, $r_i : a^{m+p_i}/a^{m+p_i-q_i} \to \pm 1(0, \{3\})$

For grammar G, activation of production rule R_i generates a single terminal symbol b. For SNPSP system Π, an activation of plasticity rule r_i generates a 3-symbol substring, either '000' or '011'. If R_i of G generates '1', the corresponding r_i of Π will generate '011'. If R_i generates '0', the corresponding r_i will generate '000'.

Π works on a 3-time step cycle.

1. Step 1: r_i will activate and will connect σ_0 to σ_3 sending a spike to σ_3. If $b = 1$ in R_i, r_i will connect σ_0 to σ_1 and σ_2 and send a spike to both neurons. $(m + p_i - q_i)$ spikes are consumed and q_i spikes remain in σ_0. At the same time, output neuron σ_1 generates '0' since no rule is activated.
2. Step 2: All synapses created in step 1 are removed. σ_3 will spike transferring its spike to neurons $\sigma_{c_1}, \dots, \sigma_{c_m}$. If $b = 1$, σ_1 will spike generating '1' and σ_2 will also spike transferring its spike to σ_1. If $b = 0$, there will be no spike in σ_1 and σ_2 and '0' is generated.
3. Step 3: $\sigma_{c_1}, \dots, \sigma_{c_m}$ will all spike sending m spikes to σ_0 which will now have $m + q_i$ spikes. If $b = 1$, σ_1 will spike generating '1'. If $b = 0$, '0' will be generated by σ_1.

For a terminal production rule $R_i : N_{p_i} \to b_i$, the corresponding r_i will have a similar form as describe above (for non-terminal productions). The only difference is that it will consume $m + p_i$ spikes.

Form of r_i:

– If $b = 1$, $r_i : a^{m+p_i}/a^{m+p_i} \to \pm 3(0, \{1, 2, 3\})$.
– If $b = 0$, $r_i : a^{m+p_i}/a^{m+p_i} \to \pm 1(0, \{3\})$

When activated, r_i will consume all $m + p_i$ spikes in σ_0. Similar to the activation of non-terminal plasticity rules, the system will eventually generate the substring $0bb$. m spikes will be sent to σ_0 by $\sigma_{c_1}, \dots, \sigma_{c_m}$. The system will halt at this point since there will be m spikes in σ_0 and no rule can be activated since the regular expression for any rule r_i is $E_i = a^{m+p_i}$ where $1 \leq p_i$.

The string s generated by Π can be transformed to the string s', generated by G, using the morphism $h_1 : B^* \to B^*$ defined as: $h_1(0) = 000$, $h_1(1) = 011$. h_1 can be extended to accomodate longer strings in a natural way.

□

3.4 SNPSP Systems for Recursively Enumerable Languages

Theorem 4. *For every alphabet $V = \{a_1, a_2, a_3, ..., a_k\}$ there is a morphism $h_1 : (V \cup \{b, c\})^* \to B^*$ and a projection $h_2 : (V \cup \{b, c\})^* \to V^*$ such that for each language $L \subseteq V^*$, $L \in RE$, there is an SNPSP system Π such that $L = h_2(h_1^{-1}(L(\Pi)))$.*

The technique for the proof of this theorem is adapted from the proof of a similar theorem (theorem 9) in [3] for SNP systems. We use the same proof technique but instead of using SNP system neurons that have forgetting rules and rules with delay (a spiking rule in SNP system can have some time delay between rule activation and sending of spikes), SNPSP system neurons and semantics of plasticity rules are used. This shows the usefulness of adding and removing synapses for programming the system.

Proof Overview. For any string $x \in V^*$, a value $val_k(x)$ in base $k+1$ can be associated with x. Base $k+1$ is used in order to consider the symbols $a_1, ...a_k$ as digits $1, ..., k$ thus avoiding the digit 0. This notation can be extended to sets of strings in a natural way. Since $L \in RE$ then $val_k(L)$ is a recursively enumerable set of numbers which means there is a deterministic register machine that can accept $val_k(L)$. Let the register machine that accepts $val_k(L)$ be M.

It was already shown that SNPSP systems can simulate register machines. The SNPSP system that will be constructed will use two subsystems that simulate register machine M and another register machine M_0. It will work in two phases: (1) the generating phase and (2) checking phase.

In the generating phase, the M_0 simulator part of the main SNPSP system and some other neurons (including the ouput neuron) will generate a binary string y which may corresponds to some string $x \in L$. i.e. $x' = h_2(h_1^{-1}(y))$ and $x = x'$. Also in generating phase, the value $val_k(x')$ is generated in one of the neurons (c_1) representing a register of M_0. Neuron c_1 is also accessible to the part of the SNPSP system simulating M. It also represents a register of machine M.

In the checking phase, part of the system will simulate the machine M whose function is to accept the value $val_k(x')$ if $x' \in L$. If $x' \in L$, then part of the SNPSP system simulating M will trigger the halting of the entire system. If $val_k(x')$ is not accepted, then $x' \notin L$ and part of system simulating M will not halt and thus no string is generated by the SNPSP system.

Proof. For a word $x \in L$, Π will generate a corresponding word $y \in B^+$. The relationship between words x and y can be established using two morphisms, h_1 and h_2.

Let $x = a_{i_1}a_{i_2}a_{i_3}...a_{i_m}$ be some word in L. The corresponding word that will be generated by Π is:

$$y = \ 10^{i_1}1 \mid 0^{j_1}1 \mid 10^{i_2}1 \mid 0^{j_2}1 \mid 10^{i_3}1 \mid 0^{j_3}1 \mid ... \mid 10^{i_m}1 \mid 0^{j_m}1$$

For each symbol a_{i_t} in word x, we can associate the two substrings $10^{i_t}1$ and $0^{j_t}1$ in y to a_{i_t} where $1 \leq t \leq m$.

We define the morphism h_1 as follows, $h_1(a_i) = 10^i1$ for $1 \leq i \leq k$, $h_1(b) = 0$, $h_1(c) = 01$. If the inverse of h_1 is applied to y, we will have the string $\overline{x} = h_1^{-1}(y)$:

$$\overline{x} = \ a_{i_1} \mid b^{j_1-1}c \mid a_{i_2} \mid b^{j_2-1}c \mid a_{i_3} \mid b^{j_3-1}c \mid ... \mid a_{i_m} \mid b^{j_m-1}c$$

Applying h_1^{-1} to y, transformed the substrings $10^{i_t}1$ in y to a_{i_t} in $\overline{x}$ and the substrings $0^{j_t}1$ in y to $b^{j_t-1}c$ in $\overline{x}$.

To transform $\overline{x}$ to x, we only need to remove symbols b and c from $\overline{x}$. We define another morphism h_2 as, $h_2(a_i) = a_i$ for $1 \leq i \leq k$, $h_2(b) = \lambda$, $h_2(c) = \lambda$. Applying h_2 to $\overline{x}$, we will have the resulting string $x = h_2(\overline{x})$.

Using the morphism h_1 and h_2, relationship between a word $x \in L$ and the word y generated by the Π is $x = h_2(h_1^{-1}(y))$.

After establishing how $x \in L$ is related to the generated word y, we now describe how x is associated with a natural number $val_k(x)$. A word $x \in V^+$ where $V = \{a_1, a_2, a_3, \ldots, a_k\}$ can be associated with a natural number denoted by $val_k(x)$. For each symbol $a_i \in V$, we assign a corresponding number i where $1 \leq i \leq k$. For a word $x = a_{i_1}a_{i_2}a_{i_3}...a_{i_m}$, we can now define $val_k(x)$ as:

$$val_k(x) = i_1 \cdot (k+1)^{m-1} + i_2 \cdot (k+1)^{m-2} + i_3 \cdot (k+1)^{m-3} + \ldots + i_m \cdot (k+1)^0$$

Example. $V = \{a, b, c, d\}$ and $x = dabbc \in V^+$. We assign 1 to a, 2 to b, 3 to c, and 4 to d. $val_4(dabbc) = 4 \cdot (5^4) + 1 \cdot (5^3) + 2 \cdot (5^2) + 2 \cdot (5^1) + 3 \cdot (5^0)$.

The SNPSP system Π generates the word y for some $x \in L$. Internally, the system is also computing the number $val_k(x)$ while y in being generated. $val_k(x)$ is computed inside the system Π iteratively. If $x = a_{i_1}a_{i_2}a_{i_3}...a_{i_m}$, the iterative process of computing $val_k(x)$ is shown below:

$$\begin{gathered} n^{(1)} = i_1 \\ n^{(2)} = n^{(1)} \cdot (k+1) + i_2 \\ n^{(3)} = n^{(2)} \cdot (k+1) + i_3 \\ \vdots \\ n^{(m)} = n^{(m-1)} \cdot (k+1) + i_m \\ val_k(x) = n^{(m)} \end{gathered}$$

In the generating phase, $val_k(x)$ is computed and part of y is generated by Π. In order to perform iterative process of computing $val_k(x)$, parts of Π will simulate a register machine M_0 whose responsiblity is computing $n^{(t)} = n^{(t-1)} \cdot (k+1) + i_t$ for $1 \leq t \leq m$. M_0 contains the two registers, c_0 and c_1. Before computing $n^{(t)}$, register c_1 contains the value $n^{(t-1)}$ while register c_0 contains i_t. After M_0 performed the computation, c_1 will now contain the value $n^{(t)}$.

Initially, both c_0 and c_1 contain the value 0. The output neuron of Π will spike generating '1' as first symbol of y and the first iteration for computing $val_k(x)$ will now begin. Per iteration, the number i_t $(1 \leq i_t \leq k)$ is non-deterministically selected and this value is generated at register c_0. Then M_0 will perform the calculation for $n^{(t)}$ using the values in c_0 and in c_1 (which is $n^{(t-1)}$).

Per iteration, for the non-deterministic selection of i_t, Π will generate the substring $0^{i_t}1$. During the computation of $n^{(t)}$, Π will generate the substring 0^{j_t}. After performing the computation of $n^{(t)}$, Π can continue generating $n^{(t+1)}$ (by selecting some i_{t+1} and computing $n^{(t+1)}$ or stop at $val_k(x) = n^{(t)}$). If Π non-deterministically decides to continue computing $n^{(t+1)}$, Π will output the substring '11' and the system will continue with another generating phase iteration $t+1$. At this point, the generated string y is:

$$y = \ 1 \mid 0^{i_1}1 \mid 0^{j_1} \mid 11 \mid 0^{i_2}1 \mid 0^{j_2} \mid 11 \mid 0^{i_3}1 \mid 0^{j_3} \mid 11 \mid ... \mid 0^{i_t}1 \mid 0^{j_t} \mid 11$$

At some iteration m, the system can non-deterministically decide to stop computing $val_k(x)$. At this point, register c_1 will contain the value $n^{(m)} = val_k(x)$. The last substring generated during the last (m^{th}) iteration of the generating phase is 0^{j_m}. In this last iteration, the substring '11' is not generated by Π.

Π will proceed with the checking phase. The checking phase involves parts of the system that simulate the register machine M. M also contains the register

c_1 which holds the value $val_k(x)$. M will decide if it will accept $val_k(x)$ or not. M is designed to accept $val_k(x)$ only if $x \in L$. If M accepts $val_k(x)$, then it will halt and so entire system Π will also halt generating the string y. Otherwise, M will not halt which means Π will not halt and no string is generated by the system.

During the checking phase, the system is continually generating the symbol '0'. It is only after M halts that Π will generate the last symbol of y which is '1'. If 0^s1 is the substring generated during checking phase (checking phase took s time steps), the generated string y of the system Π is:

$$y = \; 1 \mid 0^{i_1}1 \mid 0^{j_1} \mid 11 \mid 0^{i_2}1 \mid 0^{j_2} \mid 11 \mid 0^{i_3}1 \mid 0^{j_3} \mid 11 \mid ... \mid 0^{i_m}1 \mid 0^{j_m} \mid 0^s1$$

The entire SNPSP system Π is shown in Figure 3.

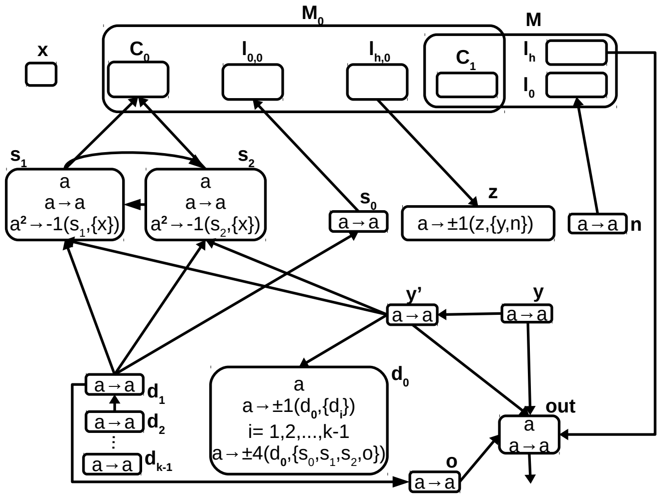


Fig. 3. SNPSP System for RE languages.

Every iteration t ($1 \leq t \leq m$) in the generating phase, neurons s_1 and s_2 will send spikes to each other and to neuron c_0. c_0 receives a number of spikes that is a multiple of 2 since register machines simulated using an SNPSP system represent a value g using $2g$ spikes.

The subsystem that contains the d_i neurons where $0 \leq i \leq k-1$, we call this *d-subsystem*, will non-deterministically decide when to stop neurons s_1 and s_2 from supplying spikes to neuron c_0 and trigger M_0's computation of $n^{(t)}$. The d-subsystem acts like a single SNP system neuron with rules with delay "$a \to a : i$".

While neurons s_1 and s_2 are supplying spikes to neuron c_0, neuron *out* is not spiking so the system is generating the substring 0^i in those time steps. After some 'delay' i non-deterministically selected by neuron d_0, the d-subsystem will send spikes to neurons s_0, s_1, s_2, and o. The spikes sent in neurons s_1 and s_2 will stop them from spiking. Having two spikes in neuron s_1 or s_2 will activate the rule of the form $a^2 \rightarrow -1(s_{1/2}, \{x\})$ that acts as forgetting rule. It consumes two spikes and tries to remove a non-existing synapse to neuron x. The spike sent to s_0 will be forwarded to the initial instruction neuron $l_{0,0}$ of M_0 and will trigger the computation of $n^{(t)}$. The spike sent to neuron o will be forwarded to neuron *out* and then to the environment. This will generate the symbol '1'. The process of non-determistically selecting i and generating $2i$ spikes in c_0 produces the substring 0^i1.

M_0 will compute $n^{(t)}$ and place its value in neuron c_1. During the computation of $n^{(t)}$, the system is generating the substring 0^j symbol. M_0's halting instruction neuron $l_{h,0}$ will send a spike to neuron z. Neuron z can non-deterministically decide to continue another iteration by sending a spike to neuron y or to go to the checking phase by sending a spike to neuron n. If neuron z decides to send a spike to neuron y, neuron y will forward the spike to neurons *out* and y'. Neuron *out* will then spike generating the symbol '1'. At the same time step, neuron y' will send a spike to neurons *out*, s_1, s_2 and d_0 it will activate neurons s_1 and s_2, the d-subsystem, and neuron *out*. Then neuron *out* will generate another '1' symbol. The process of computing $n^{(t)}$ and continuing another iteration of the generating phase produces the substring 0^j11.

If neuron z decides to go to the checking phase, sending a spike to neuron n, neuron n will forward the spike to the initial instruction neuron l_0 of machine M. Machine M will determine if it will accept the value in c_1 (generated $val_k(x)$). During M's decision process, the system is generating the substring 0^s. If M halts then the value in c_1 is accepted and M's halt instruction neuron l_h will send a spike to neuron *out* which will then generated the last symbol '1'. If the value in c_1 is not accepted, then M will not halt which means Π will not halt and no string is be generated. If the checking phase halts, the substring produced in the checking phase is 0^s1.

□

3.5 Discussions

There are a lot literature that study how languages can be generated using the different variants of the SNP system. Most of them use similar techniques to construct the systems for generating languages in FIN, REG, RE. e.g. (1) the idea of simulating right-linear regular for REG and (2) simulating register machines for RE. Those techniques are also used in this paper. It interesting to see how the different rule types (spiking, forgetting, extended spiking, plasticity) of the different SNP variants are used to construct the systems for generating languages.

In devising the SNPSP construction procedures, we made some observations about SNPSP systems. Using a neuron chain, as shown in the proof in section

3.1,a spiking delay can be simulated in SNPSP system. The main difference with the SNPSP system delay to the delay using a spiking rule in classic SNP system is that no neuron in the SNPSP system is *closed* when the spike propagates through the neuron chain (simulating delay).

Another observation is that a forgetting rule in classic SNP systems can easily be simulated in SNPSP systems using plasticity rules that will perform no action aside from spike consumption. i.e. a plasticity rule that attempts to delete non-existing synapses or a plasticity rule that attempts to create already existing synapses.

These observations were used to adapt the proof of a theorem (theorem 9) in [3] for SNP system and use it to prove theorem 4 for SNPSP system. The proof in [3] uses rules with delay and forgetting rules. Rules with delay and forgetting rules are simulated in SNPSP system using neuron chains and plasticity rules.

There is a main advantage for having plasticity rules as compared to having only spiking rules with delay or extended spiking rules with delay. It is the ease of communicating a neuron's choice/decision to other neurons using a plasticity rule. i.e. Neuron x, using a plasticity rule, can choose to connect to either neuron a, neuron b, or neuron c thus 'selecting' a neuron. Using only extended rules, neuron x can communicate a choice/decision to neurons a, b, c by using multiple extended rules each of the rules having the same prerequisites (E/a^c) but either having different delays (d) or different number of produced spikes (a^p). Neurons a, b, c will all receive the same number of spikes at the same time. The only way for them to determine which of them is 'selected' by neuron x is for each of them expect a certain amount of spikes at specific times. If one of the neurons a, b, c receives the amount of spikes it was expecting at the time it was expecting it, then that neuron knows that it was selected by neuron x. The other neurons will receive the same amount of spikes at that same time, but they will not be expecting that amount of spikes at that time and they know that they were not selected by neuron x. This ease of communicating a choice/decision by selecting a neuron or as set of neuron is shown in the systems constructed for FIN (theorem 1), REG and RE.

Having plasticity rules can reduce the number of neurons in the system. It can eliminate the extra neurons needed for one neuron to communicate its choice/decision to other neurons.

A P system known as *extended spiking neural P (ESNP) system* introduced in [1] uses a generalized form of rules that already contains the extended spiking rule and plasticity rule as special cases. When activated, this general rule can send different amount of spikes at different target neurons. A delay in the rule firing can also be specified. After rule activation, the rule will only send the spikes after the specified time delay. An additional feature of this rule is *transition delay* on the synapses(axons). After some time the rule will fire sending spikes to the target neurons, the spikes will only reach the target neurons after the specified transition delay. It's possible that each of the synapses to the target neurons will have different transition delays.

4 Final Remarks

In this work, we presented SNPSP systems as language generators. Specifically, we showed two procedures for constructing SNPSP systems for generating FIN languages, one procedure for constructing SNPSP systems for REG languages, and one procedure for constructing SNPSP systems for RE languages. The procedures show that FIN, REG and RE languages can be represented using SNPSP systems. It is a natural question to ask whether the other families in the Chomsky hierarchy, context-free and context-sensitive languages, can be represented using SNPSP systems.

Sections 3.1 and 3.2 show two different ways of representing FIN languages using SNPSP systems. It is likely that there are also multiple ways to represent REG and RE languages using SNPSP systems. It might be interesting to create multiple representations of the families of languages using SNPSP systems and compare the different representations to each other. One might also devise criteria to assess in which situations will one representation be considered 'better' than the other representations.

With regards to language generation, one can also characterize the families of languages in the Chomsky hierarchy using SNPSP systems. By characterization of a family of languages (e.g. REG) using SNPSP systems, we mean finding a class of SNPSP systems whose set of generated languages is exactly the family in the Chomsky hierarchy. Characterization results will be 'stronger' results compared to SNPSP system representations. Having a characterization result will give additional information about the class of SNPSP systems used to characterize the family of languages. i.e. A representation result of REG can say that given a language $L \in REG$, there is an SNPSP system Π that generates L or a language L' that corresponds (via morphisms) to L. A characterization result of REG can also tell the additional information that given an SNPSP system Π in the class (SNPSP system class used for REG characterization) then the language generated by Π can also be generated by regular grammar.

Acknowledgements

F.G.C. Cabarle is grateful for a Faculty Research Incentive Award (2016–2017) from the College of Engineering, UP Diliman, the Project No. 161606 PhDIA and an RLC grant 2016–2017, both from the UP Diliman, OVCRD, and the Soy and Tess Medina Professorial Chair Award for 2016–2017. F.G.C. Cabarle and X. Zeng are supported by the National Natural Science Foundation of China (No. 61672033) and Xiamen University research fund 20720170054.

References

1. Alhazov, A., Freund, R., Oswald, M., Slavkovik, M.: Extended Spiking Neural P Systems. In: Membrane Computing, pp. 123–134. Springer Berlin Heidelberg (2006)

2. Cabarle, F.G.C., Adorna, H.N., Pérez-Jiménez, M.J., Song, T.: Spiking Neural P Systems with Structural Plasticity. Neural Computing and Applications 26(8), 1905–1917 (2015)
3. Chen, H., Freund, R., Ionescu, M., Păun, Gh., Pérez-Jiménez, M.J.: On String Languages Generated by Spiking Neural P Systems. Fundam. Inform. 75(1-4), 141–162 (2007)
4. Chen, H., Ionescu, M., Ishdorj, T.O., Păun, A., Păun, Gh., Pérez-Jiménez, M.J.: Spiking Neural P Systems with Extended Rules: Universality and Languages. Natural Computing 7(2), 147–166 (2008)
5. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking Neural P Systems. Fundam. Inf. 71(2,3), 279–308 (Feb 2006)
6. Kong, Y., Zhang, Z., Liu, Y.: On String Languages Generated by Spiking Neural P Systems with Astrocytes, pp. 225–229. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
7. Krithivasan, K., Metta, V.P., Garg, D.: On String Languages Generated by Spiking Neural P Systems with Anti-Spikes. International Journal of Foundations of Computer Science 22(01), 15–27 (2011)
8. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1967)
9. Pan, L., Păun, Gh.: Spiking Neural P systems with Anti-Spikes. International Journal of Computers, Communications and Control 4(3), 273–282 (2009), cited By 81
10. Pan, L., Păun, Gh., Pérez-Jiménez, M.J.: Spiking Neural P Systems with Neuron Division and Budding. Science China Information Sciences 54(8), 1596 (2011)
11. Pan, L., Wang, J., Hoogeboom, H.J.: Spiking Neural P Systems with Astrocytes. Neural Computation 24(3), 805–825 (mar 2012)
12. Păun, Gh., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., New York, NY, USA (2010)

Bi-simulation Between P Colonies and P Systems with Multi-stable Catalysts

Erzsébet Csuhaj-Varjú[1] and Sergey Verlan[2]

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

[2] Université Paris Est, LACL (EA 4219), UPEC, F-94010, Créteil, France
`verlan@u-pec.fr`

Abstract. The general concept, called the formal framework of P systems provides a representation to study and analyze different models of P systems. In this paper, two well-known models, P colonies and P systems with multi-stable catalysts are considered. We show that the obtained representations are identical, thus both models can be related using a bi-simulation. This fact opens new approaches both for studying P colonies and catalytic P systems.

1 Introduction

Due to different motivations, there have been several variants of P systems introduced. However, all models have some common basic features as summarized in [5, 9]. Among these characteristics we find

- a description of the initial structure or architecture (indicating the graph relation between the compartments and any additional information as labels, charges, etc.),
- a list of the initial multisets of objects present in each compartment at the beginning of the computation,
- a set of rules, acting over objects and / or over the structure.

Usually, the configuration of a P system is represented by the current contents of the compartments and the current structure of the system.

P systems work with transitions between configurations; a finite sequence of such transitions of a P system Π starting with the initial configuration and ending in some final configuration is called a computation. The final configuration is usually given by halting.

To give a more precise description of the semantics, the following notions (functions) were defined:

- *Applicable*$(\Pi, \mathcal{C}, \delta)$ – the set of multisets of rules of Π applicable to the configuration $\mathcal{C}$, according to some derivation mode δ.

- $Apply(\Pi, \mathcal{C}, R)$ – the configuration obtained by the (usually parallel) application of the multiset of rules R to the configuration $\mathcal{C}$.
- $Halt(\Pi, \mathcal{C}, \delta)$ – a predicate that yields true if $\mathcal{C}$ is a halting configuration of the system Π using the derivation mode δ.
- $Result(\Pi, \mathcal{C})$ – a function giving the result of the computation of the P system Π when the halting configuration $\mathcal{C}$ has been reached. Usually, this is an integer function. However, generalizations as for example, Boolean or vector functions can also be considered.

We note that δ, above, differs from the dissolution symbol used in some P system models.

The transition of a P system Π according to the derivation mode δ (usually, the maximally parallel derivation mode) is defined as follows: the system changes from a configuration $\mathcal{C}$ to $\mathcal{C}'$ (written as $\mathcal{C} \Rightarrow \mathcal{C}'$) iff

$$\mathcal{C}' = Apply(\Pi, \mathcal{C}, R), \text{ for some } R \in Applicable(\Pi, \mathcal{C}, \delta)$$

The result of the computation of a P system is usually interpreted as the union of the results of all possible computations.

The precise interpretation of the four notions (functions) above depends on the chosen model of P systems. The goal of works [4, 5, 9] was to provide a concrete family of P systems based on the structure of *network of cells* together with a series of definitions of the functions above. The obtained model as well as the accompanying tools and methods together are called the *formal framework of P systems.* It has the property that most of the existing models of P systems could be obtained by a strong bi-simulation of a restricted version (eventually, using a simple encoding) of this formal framework with respect to different parameters, see [10] for some examples. We recall that a simulation of one transitional system by another corresponds to an order relation on corresponding equivalent states [6]. Basically, this means that a step in the simulated system corresponds to one or several steps in the simulating one. In the case of a strong simulation, one step of the simulated system is performed using one step in the simulating system. If two systems can simulate each other, then we speak about bi-simulation.

In this paper, based on formal framework we provide a strong bi-simulation between two well-known models, namely P colonies and multi-stable (purely) catalytic P systems. P colonies are a finite collection of agents which interact with a shared environment via their own sets of programs. Each program is a limited number of very simple rules. Under functioning, the agents act in a maximally parallel manner and they change their own state and exchange symbols with the environment. Purely catalytic P systems are given with multiset rewriting rules where each rule has occurrences of distinguished symbols called catalysts. In the original model, catalysts cannot change, in case of multi-stable catalytic P systems catalysts are allowed to change only to some other, distinguished catalysts.

Our bi-simulation demonstrates that although the two models are formally different, one can be used to solve problems concerning the other one. For example, both models are computationally complete, thus a proof for one of the models can be ”translated” to a proof for the other one.

After providing the bi-simulation and some examples, we discuss the results and propose topics for future research.

2 Definitions and Notations

We assume that the reader is familiar with basic notions of formal language theory and membrane computing; for further details consult [7] and [8].

For a finite multiset of symbols M over an alphabet V, $supp(M)$ denotes the set of symbols in M (the support of M) and $|M|$ denotes its size, i.e., the total number of its symbols. By $|M|_x$, the number of occurrences of symbol x in M is denoted. By V° we denote the set of all finite multisets over V.

Throughout the paper, every finite multiset M is given as a string w, where M and w have the same number of occurrences of symbol a, for each $a \in V$.

2.1 Network of Cells

In this section we provide a summarized version of the definition of a *network of cells*, the class containing all networks of cells forming the structure of the formal framework. The definitions are based on those given in [5]. This version considers only static P systems where the membrane structure does not change under the computation (this also includes systems with the dissolution of membranes). We note that in [4], an extension of the formal framework to P systems with dynamically evolving structure is proposed. However, in order to have a more simple presentation, in this paper we will only consider the first variant. We remark that in the case of static structures both variants coincide, although the notation is slightly different.

Definition 1 ([5]). *A* network of cells of degree $n \geq 1$ *is a construct*

$$\Pi = (n, V, w, Inf, R)$$

where

1. *n is the number of cells;*
2. *V is an* alphabet*;*
3. *$w = (w_1, \dots, w_n)$ where $w_i \in V^\circ$, for all $1 \leq i \leq n$, is the* finite multiset initially associated to cell *i;*
4. *$Inf = (Inf_1, \dots, Inf_n)$ where $Inf_i \subseteq V$, for all $1 \leq i \leq n$, is the* set of symbols occurring infinitely often in cell *i (in most of the cases, only one cell, called the* environment*, will contain symbols occurring with infinite multiplicity);*
5. *R is a finite set of rules of the form*

$$(X \to Y; P, Q)$$

where $X = (x_1, \ldots, x_n)$, $Y = (y_1, \ldots, y_n)$, $x_i, y_i \in V^\circ$, $1 \leq i \leq n$, are vectors of multisets over V and $P = (p_1, \ldots, p_n)$, $Q = (q_1, \ldots, q_n)$, p_i, q_i, $1 \leq i \leq n$ are finite sets of multisets over V. We will also use the notation

$$(1, x_1) \ldots (n, x_n) \rightarrow (1, y_1) \ldots (n, y_n)\,; [(1, p_1) \ldots (1, p_n)]; [(1, q_1) \ldots (n, q_n)]$$

for a rule $(X \rightarrow Y; P, Q)$; moreover, if some p_i or q_i is an empty set or some x_i or y_i is equal to the empty multiset, $1 \leq i \leq n$, then we may omit it from the specification of the rule.

The semantics of the above rule is as follows: objects x_i from cells i are rewritten into objects y_j in cells j, $1 \leq i, j \leq n$, if every cell k, $1 \leq k \leq n$, contains all multisets from p_k and does not contain any multiset from q_k. In other words, the first part of the rule specifies the rewriting of symbols, the second part of the rule specifies permitting conditions and the third part of the rule specifies the forbidding conditions.

For a rule r of the form above, the set

$$\{i \mid x_i \neq \lambda \text{ or } y_i \neq \lambda \text{ or } p_i \neq \emptyset \text{ or } q_i \neq \emptyset\}$$

induces a (hypergraph) relation between the interacting cells. However, this relation does not need to give rise to a *structure* relation like a tree as in P systems or a graph as in tissue P systems.

A *configuration* C of Π is an n-tuple of multisets over V $(u_1, \ldots, u_n)$ satisfying $u_i \cap Inf_i = \emptyset$, $1 \leq i \leq n$.

In the sequel, networks of cells as intermediate models will assist to establish a bi-simulation between two variants of P systems, namely P colonies and P systems with multi-stable catalysts.

2.2 P Colonies

Next we provide the concept of a P colony, based on the formalism given in [7].

A P colony $\Pi = (O, e, f, C_1, \ldots, C_n)$, consists of n cells (agents) C_i, $1 \leq i \leq n$, each of them consisting of a multiset of exactly k symbols and an environment consisting of initially a distinguished symbol e in an unbounded number of copies. Every cell C_i has a set of programs $\{p_{i,1}, \ldots, p_{i,k_i}\}$, where each program $p_{i,j}$ consists of exactly k rules of the forms $a \rightarrow b$ (*evolution rule* or *internal point mutation*), $c \leftrightarrow d$ (*one object exchange* with the environment), or r_1/r_2 (*priority rule*, where r_1 and r_2 are arbitrary combinations of point mutation and/or exchange rules).

The computation starts in the initial configuration, i.e., the n-tuple of the initial contents of the cells. It can be performed in the maximally parallel (*par*) or in the sequential (*seq*) mode, the computation mode is assigned to the system at the beginning. If no more program is applicable, then the P colony halts and the result is collected as the number of distinguished symbols f in the environment. The result of the computation of Π is denoted by $N(\Pi)$.

We note that the result can be defined in such a way, too, that we consider the number of all symbols in the environment which are different from e.

The number of cells, the maximal number of programs in a cell, and the maximal number of rules in each program in a given P colony Π are called the degree, the height, and the capacity of Π, respectively.

The family of sets of numbers computed in the derivation mode x for $x \in \{par, seq\}$ by P colonies of capacity k, degree at most $n \geq 1$ and height at most $h \geq 1$, without (resp. with) using priority rules in their programs, is denoted by $NPCol_x(k, n, h)$ (resp. $NPCol_xK(k, n, h)$).

Notice that a strong bi-simulation of the P colony model and the formal framework can be given as follows.

– each rule $a \to b$ in p_{ij} becomes $r_{ij} : (i, a) \to (i, b)$;
– each rule $a \leftrightarrow b$ in p_{ij} becomes $r_{ij} : (i, a)(0, b) \to (i, b)(0, a)$;
– each rule r_1/r_2 in p_{ij} becomes:
  – $p^1_{ij} : r_1$, $p^2_{ij} : r_2; [\emptyset]; [\{(i, a)\}]$ if r_1 is an evolution rule ($a \to b$)
  – $p^1_{ij} : r_1$, $p^2_{ij} : r_2; [\emptyset]; [\{(i, a)(0, b)\}]$ if r_1 is an exchange rule ($a \leftrightarrow b$).

For the derivation mode, each program becomes a rule partition and then the derivation mode requires to be maximal, but using exactly k rules from each partition (or using all rules from a partition). In the sequential case, the derivation mode prescribes to use only one partition (but all rules from that partition).

Example 1 ([10]). Consider the following P colony Π having 3 cells. For simplicity, we provide only the initial multisets and the programs of the cells.

– C_1 contains the initial multiset aa and the following programs: $p_{11} : a \to b, a \leftrightarrow e$, $p_{12} : a \to c, a \leftrightarrow e$, $p_{13} : b \to a, e \to a$.
– C_2 contains the initial multiset be and the following program: $p_{21} : b \leftrightarrow e, e \to b$.
– C_3 contains the initial multiset ee and the following programs: $p_{31} : e \leftrightarrow a, e \leftrightarrow b$, $p_{32} : b \to f, a \to b$, $p_{33} : f \leftrightarrow a, b \to b$.

Figure 1 shows a graphical representation of this system.

We transform this system to a network of cells Π' having 4 cells (numbered from 0 to 3). Cell 0 corresponds to the environment. Cells 1, 2, 3 correspond to the cells of Π and have the same initial contents as the corresponding agent. We define $Inf_0 = \{e\}$. System Π' contains the following rules:

Rules simulating programs from the first cell:

$r_{111} : (1, a) \to (1, b)$ $r_{112} : (1, a)(0, e) \to (1, e)(0, a)$
$r_{121} : (1, a) \to (1, c)$ $r_{122} : (1, a)(0, e) \to (1, e)(0, a)$
$r_{131} : (1, b) \to (1, a)$ $r_{132} : (1, e) \to (1, a)$

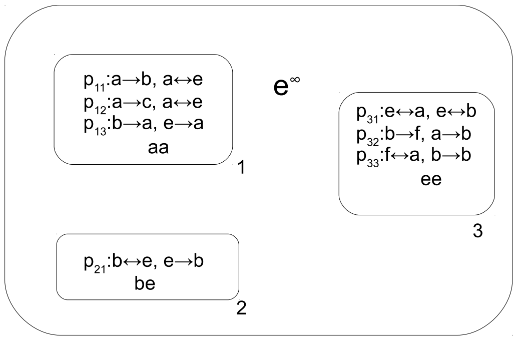


Fig. 1. The P colony from Example 1.

Rules simulating programs from the second cell:

$$r_{211} : (2, b)(0, e) \to (2, e)(0, b) \qquad r_{212} : (2, e) \to (2, b)$$

Rules simulating programs from the third cell:

$$\begin{array}{ll} r_{311} : (3, e)(0, a) \to (3, a)(0, e) & r_{312} : (3, e)(0, b) \to (3, b)(0, e) \\ r_{321} : (3, b) \to (3, f) & r_{322} : (3, a) \to (3, b) \\ r_{331} : (3, f)(0, a) \to (3, a)(0, f) & r_{332} : (3, b) \to (3, b) \end{array}$$

We remark that the derivation mode of P colonies groups rules corresponding to programs, uses maximal parallelism or sequential mode, and it requires that all rules from a group should be used. Since working with one symbol, the group r_{111} and r_{112} from the above example is equivalent to the application of a single rule $r_{11} : (1, aa)(0, e) \to (1, be)(0, a)$. Hence, we obtain that a program corresponds to a more complicated rule, and k is the size of the left-hand side (LSH) of this rule (and equal to the right-hand side, i.e., RHS). By considering such rules, the evolution of a P colony becomes just maximally parallel or sequential.

This consideration yields to the following network of cells Π'' (working in sequential or maximally-parallel manner):

$$\begin{array}{ll} r_{11} : (1, aa)(0, e) \to (1, be)(0, a) & r_{12} : (1, aa)(0, e) \to (1, ce)(0, a) \\ r_{13} : (1, be) \to (1, aa) & \\ r_{21} : (2, be)(0, e) \to (2, be)(0, b) & \\ r_{31} : (3, ee)(0, ab) \to (3, ab)(0, ee) & r_{32} : (3, ab) \to (3, fb) \\ r_{33} : (3, bf)(0, a) \to (3, ab)(0, f) & \end{array}$$

Since the number of combinations of objects in an agent is finite, it can be represented by a single symbol, a state. Also, symbol e from cell 0 can be ignored as it carries no information. This permits to deduce that a P colony corresponds to a cooperative rewriting mechanism with the size of LHS or RHS at most $k+1$

and forbidding conditions (if checking rules are present). In the next section we refine this observation by showing that the rewriting is performed in a catalytic-like manner.

2.3 P systems with Multi-stable Catalysts

In this section we extend the notion of a P system with catalysts to that variant where the catalysts can have multiple states. For catalytic P systems, consult [7].

Let V and C be two disjoint alphabets, let $k > 0$, and let C have a partition $C = C_1 \cup \cdots \cup C_n$ such that $1 \leq |C_i| \leq k$. We say that each partition is a multi-stable catalyst and we define $Period(C_i) = |C_i|$ the *period* of the catalyst C_i, $1 \leq i \leq n$.

In the sequel, the elements of a multi-stable catalyst C_i having period k will be denoted by $c_i^{(j)}$, $1 \leq j \leq k$.

A k-states multi-stable (purely) catalytic P system with n catalysts is a construct $\mathit{\Gamma} = (V, C, R, w)$, where V is the set of non-catalytic objects of $\mathit{\Gamma}$, $C = C_1 \cup \cdots \cup C_n$ with catalysts C_i, having period at most k, $1 \leq i \leq n$.

R is a finite set of rules where each rule is of the following form

$$c_i^{(j)}u \rightarrow c_i^{(t)}v, \text{ where } 1 \leq j, t \leq Period(C_i), 1 \leq i \leq n \text{ and } u, v \in V^{\circ}.$$

The initial configuration of $\mathit{\Gamma}$, w is a multiset over $V \cup C$, with at most one element of each multi-stable catalyst C_i, i.e., $w \subseteq (V \cup C)^{\circ}$, with the condition that $\sum_{j=1}^{Period(C_i)} |w|_{c_i^{(j)}} \leq 1$, $1 \leq i \leq n$.

Notice that the rules of a multi-stable catalytic P system with multiple states can easily be represented in the formal framework by [5],[10] as follows:

$$(0, c_i^{(j)}u) \rightarrow (0, c_i^{(t)}v), \text{ for all } c_i^{(j)}u \rightarrow c_i^{(t)}v \in R.$$

As standard P systems, the k-states multi-stable (purely) catalytic P systems $\mathit{\Gamma}$ with n catalysts work by transitions of their configurations where the rules are applied in the maximally parallel manner. A successful computation performed by $\mathit{\Gamma}$ is a finite sequence of transitions starting in its initial configuration and ending by halting; the result of the computation is the number of non-catalytic objects in the halting configuration. The result of the computation is denoted by $N(\mathit{\Gamma})$.

3 Bi-simulation of the Two Models

In this section we demonstrate the equivalence of P colonies and multi-stable (purely) catalytic P systems by using their representation in the above formal framework.

We first show that any (recursively enumerable) set of numbers that can be computed by a P colony (in the sense defined above) can be computed by a multi-stable catalytic P system as well.

Theorem 1. *For any P colony $\Pi = (O, e, w_0, P_1, \ldots, P_n)$ of size (k, n, h) there exists a h'-states multi-stable purely catalytic P system $\Gamma = (O, C, w, R)$ with n catalysts with $h' \leq h + 1$ such that $N(\Pi) = N(\Gamma)$.*

Proof. To simplify the presentation, we consider P colonies that do not contain checking rules.

According to the discussion above (see also [10]), every P colony can be represented by the formal framework. To this goal, any program p located in cell i is replaced by a rule of the corresponding network of cells. Let $p = p_1 \cup p_2$, where p_c contains all the communication rules and p_r contains all the rewriting rules of p. Let $lhs_c(p)$ (resp. $rhs_c(p)$) denote the multiset of letters of all left-hand (resp. right-hand) sides of the communication rules; we consider the same notation for the rewriting rules, using the index r. For simplicity, we will speak of sum of the left-hand sides (resp. right-hand sides) of the rules in the sequel and we will use notation $+$.

Since the definition of the P colony requires that if a program is used, then all of its rules should be applied, therefore we obtain that the execution of a program p is equivalent to the following rule given in terms of the formal framework:

$$\begin{aligned} &(i, x)(0, y) \to (i, x')(0, y'), \text{ where} \\ &\qquad x = lhs_c(p) + lhs_r(p), y = rhs_c(p), \\ &\qquad x' = rhs_c(p) + rhs_r(p), y' = lhs_c(p). \end{aligned} \tag{1}$$

Since in every step of the computation every cell in a P colony contains a constant number of objects equal to its capacity k, each cell contents can be interpreted as a number z in base $k + 1$ having exactly $|O|$ bits. Alphabet O is equal to $\{o_1, \ldots, o_s\}$ and any o_i (and e) can appear in any contents in at most k copies. Thus $|O|$ bits represent the number of occurrences of object o_i in a cell contents c. Under this interpretation, the rules of a program specify some other number z' equal to the value of the contents of the cell after the application of the program. Since the number of rules in a program is exactly k, for each number z and program p there is exactly one number z' associated.

We remark that for a cell i having h programs there are at most $h+1$ different possible configurations of the cell contents. We number these configurations from 1 to i_h, where $i_h \leq h + 1$. Let f be a bijection between all possible values of cell configurations and $1, \ldots, i_h$. Thus, we can rewrite 1 as follows:

$$\begin{aligned} &(i, c^{(f(z))})(0, y) \to (i, c^{(f(z'))})(0, y'), \text{ where} \\ &\qquad y = rhs_c(p), y' = lhs_c(p), 1 \leq z, z' \leq i_h. \end{aligned} \tag{2}$$

We can further transform this rule as follows:

$$(0, c_i^{(f(z))} y) \to (0, c_i^{(f(z'))} y') \tag{3}$$

It can clearly be seen that this rule corresponds to a rule of a multi-stable (purely) catalytic P system.

Hence, starting from the P colony Π, components of Γ can be constructed. We first remark that object e in P colonies act as an empty symbol, so we replace all its occurrences by λ in the obtained catalytic rules. First, the initial multiset of Γ is determined from the initial configuration of Π. Since every rule $c_i^{(f(z))}y \to c_i^{(f(z'))}y'$ of Γ correspond to the application of a program p in Π described above, it can easily be seen that any transition from configuration c_1 to configuration c_2 of Π corresponds to the application of an m-tuple of rules in Γ, where $m \leq n$. Notice that depending on the applicability of their programs, some components may remain inactive. Since the initial multiset of Γ contains at most n catalysts, Γ has only catalytic rules, at any computation step as many catalytic rules are applied in parallel as possible, i.e. at most n. Since these rules correspond to programs of pairwise different components of Π, every computation in Γ corresponds to a computation in Π as well. Thus, it is easy to see that the number of non-catalytic objects at halting of Γ is equal to the number of objects in the environment of Π which are different from e at halting. □

Example 2. Let us consider P colony Π from Example 1. We recall the corresponding rules.

- C_1 contains the initial multiset aa and the following programs:
  $p_{11} : a \to b, a \leftrightarrow e$, $p_{12} : a \to c, a \leftrightarrow e$, $p_{13} : b \to a, e \to a$.
- C_2 contains the initial multiset be and the following program:
  $p_{21} : b \leftrightarrow e, e \to b$.
- C_3 contains the initial multiset ee and the following programs:
  $p_{31} : e \leftrightarrow a, e \leftrightarrow b$, $p_{32} : b \to f, a \to b$, $p_{33} : f \leftrightarrow a, b \to b$.

Let $O = \{a, b, c, e, f\}$, and let $o_1 = a,\ldots,o_5 = f$, in this order. The different cell contents are $A = (aa, be, ce, ee, ab, bf)$ which correspond to numbers 00002, 01010, 01100, 02000, 00011, 010010 in base $k + 1 = 6$. For simplicity, let us denote these numbers by s_1, s_2, s_3, s_4, s_5, and s_6, respectively.

Then by constructing the multi-stable catalytic P system we obtain the following rules:

- $C_1^{s_1} \to C_1^{s_1}a$, , $C_1^{s_1} \to C_1^{s_3}a$, $C_1^{s_2} \to C_1^{s_1}$,
- $C_2^{s_2} \to C_2^{s_2}b$,
- $C_3^{s_4}ab \to C_3^{s_5}$, $C_3^{s_5} \to C_3^{s_6}$, $C_3^{s_6}a \to C_3^{s_5}f$.

Next we show that the sets of numbers computed by multi-stable catalytic P systems can be computed by P colonies as well.

Theorem 2. *For any h-states multi-stable catalytic P system $\Gamma = (O, C, w, R)$ with n catalysts there exists a P colony $\Pi = (O, e, w_0, P_1, \ldots, P_n)$ of size (k, n, h) such that $N(\Pi) = N(\Gamma)$ holds.*

Proof. We construct Π as follows. P colony Π has n cells and each cell i has $Period(C_i)$ programs. Now we will show how these programs are constructed.

Consider a rule $c_i^j u \to c_i^t v \in R$. We suppose that $|u| = |v|$. If this is not the case, then we complement the smaller multiset by adding the needed amount of symbols e. That is, if $|u| < |v|$ then let $u' = u + e^{|v|-|u|}$. Suppose that $u = u_1 \dots u_s$ and $v = v_1 \dots v_s$. We will construct the program p_j corresponding to this rule. It will be composed from two parts. The first part will contain communication rules that simulate the rewriting of u to v in the above rule. The second part contains rewriting rules that allow to complement the encoding of the catalyst state by the contents of the cell.

In order to determine the corresponding rewriting rules we should first find an encoding for each state of the catalyst. This encoding can be obtained as a solution of the following integer optimization problem.

$$\begin{aligned}
&k \to min,\\
&\sum_{a\in O} x_a^{i,j} = k, \quad 1 \le j \le Period(C_i),\\
&x_a^{i,j} \ge |v|_a, \qquad x_a^{i,t} \ge |u|_a, \qquad \text{for any } c_i^j u \to c_i^t v \in R,\\
&x_a^{i,j} \in \mathbb{N}, \quad a \in O, 1 \le i \le n, 1 \le j \le Period(C_i).
\end{aligned} \tag{4}$$

The inequalities state that the symbols that are sent out (resp. received in) by the exchange rules of the P colony belong to the coding of state j (resp. t) of the catalyst C_i.

We remark that since inequalities 4 do not impose an upper bound value for $x_a^{i,j}$, there is always a solution for this system. In case of several possible solutions, we prefer solutions having the maximal number of symbols e.

The capacity of the P colony is the value k.

Let $x_a^{i,j}$, $a \in O$, $1 \le j \le Period(C_i)$ be a solution of the above problem. Let $Code(c_i^j) = \sum_{a\in O} a^{x_a^{i,j}}$. Let $c_i^j u \to c_i^t v \in R$ and let $|u| = |v| = s$, $d^j = Code(c_i^j) - v$ and $d^t = Code(c_i^t) - u$. Suppose that $d^l = d_1^l \dots d_m^l, l \in \{j, t\}$. Then

$$p_j = (v_1 \leftrightarrow u_1; \dots v_s \leftrightarrow u_s; d_1^j \to d_1^t; \dots d_m^j \to d_m^t).$$

Now we are able to construct the colony: for every C_i, $1 \le i \le n$, P colony Π will have component P_i. The programs belonging to P_i are obtained from the rules $C_i^j u \to C_i^t v$, in the above described manner. Notice that any program of Π, determined above encodes the application of the corresponding catalytic rule, thus, the programs to be applied and the catalytic rules correspond to each other. Since at the beginning of the computation the initial state of Γ contains at most one element of each multi-stable catalyst and both systems apply the maximally parallel computation, we obtain that the two systems compute the same set of numbers. □

Example 3. To demonstrate the previous construction, we add an example.

Consider the following multi-stable catalytic P system $\Pi = (O, C, w_1, R_1)$.

$$\begin{array}{lll}
1.1: C_1^1 a \to C_1^2 bc & 2.1: C_2^1 \to C_2^2 & 3.1: C_3^1 \to C_3^2 a \\
1.2: C_1^1 a \to C_1^3 c & 2.2: C_2^2 b \to C_2^2 & 3.2: C_3^2 c \to C_3^1 b \\
1.3: C_1^2 ac \to C_1^1 aa & & 3.3: C_3^1 \to C_3^3
\end{array}$$

We transform rules 1.1, 2.2 and 3.1 by adding symbol e in order to balance the number of symbols at both sides:

$$1.1: C_1^1 ae \to C_1^2 bc \qquad 2.2: C_2^2 b \to C_2^2 e \qquad 3.1: C_3^1 \to C_3^2 a$$

Then the corresponding minimization problem is the following:

$$\begin{array}{l}
k \to min \\
x_a^{i,j} + x_b^{i,j} + x_c^{i,j} + x_e^{i,j} = k, \quad 1 \le i,j \le 3 \\
x_b^{1,1} \ge 1,\ x_c^{1,1} \ge 1,\ x_a^{1,2} \ge 1,\ x_e^{1,2} \ge 1, \\
x_c^{1,1} \ge 1,\ x_a^{1,3} \ge 1, \\
x_a^{1,1} \ge 1,\ x_c^{1,1} \ge 1,\ x_a^{1,2} \ge 2 \\
x_e^{2,2} \ge 1,\ x_b^{2,2} \ge 1 \\
x_a^{3,1} \ge 1,\ x_e^{3,2} \ge 1,\ x_c^{3,1} \ge 1,\ x_b^{3,2} \ge 1 \\
x_a^{i,j} \in \mathbb{N}, \quad a \in O, 1 \le i,j \le 3.
\end{array}$$

We can regroup inequalities by the corresponding state:

$$\begin{array}{l}
k \to min \\
x_a^{i,j} + x_b^{i,j} + x_c^{i,j} + x_e^{i,j} = k, \quad 1 \le i,j \le 3 \\
x_a^{1,1} \ge 1,\ x_b^{1,1} \ge 1,\ x_c^{1,1} \ge 1 \\
x_a^{1,2} \ge 2,\ x_e^{1,2} \ge 1 \\
x_a^{1,3} \ge 1 \\
x_b^{2,2} \ge 1,\ x_e^{2,2} \ge 1 \\
x_a^{3,1} \ge 1,\ x_c^{3,1} \ge 1 \\
x_b^{3,2} \ge 1,\ x_e^{3,2} \ge 1 \\
x_a^{i,j} \in \mathbb{N}, \quad a \in O, 1 \le i,j \le 3.
\end{array}$$

The minimal value of k is equal to 3 and one of possible solutions yields to the following codes for c_i^j:

x	c_1^1	c_1^2	c_1^3	c_2^1	c_2^2	c_3^1	c_3^2	c_3^3
$Code(x)$	abc	aae	aaa	eee	bee	ace	bee	eee

We remark that catalysts C_2 and C_3 can be represented only using two symbols.

The obtained P colony is shown in Fig. 3.

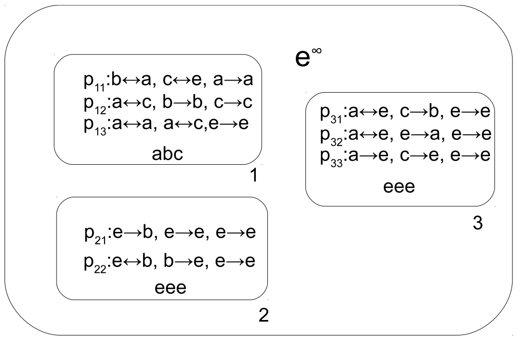


Fig. 2. The P colony constructed in Example 3.

4 Conclusions

In this paper we have shown a strong bi-simulation between the model of P colonies and pure multi-stable catalytic P systems. This result was obtained by using the formal framework for P systems as intermediate step.

As immediate consequence of the results of this paper, it is possible to rewrite existing results from the area of P colonies in terms of multi-stable catalytic P systems and conversely. These investigations are topics of future research. Another consequence is the possibility to conduct proofs in terms of purely catalytic P systems (that tend to be simpler) and automatically transform them to P colonies.

Furthermore, this article allows to establish the correspondence between different extensions of P colonies (see [3]) and particular variants of catalytic P systems. For example, the evolving environment extension [2] corresponds to the same multi-stable catalytic P systems, so it can be simulated by a P colony with a greater capacity. Homogeneous P colonies [1] correspond to catalytic P systems having same rules for all catalysts.

Other possible extensions can also be discussed. For example, non-pure catalytic systems would correspond to P colonies having special rules allowing to evolve objects by themselves in the environment. Another possibility that follows from our constructions is to consider P colonies where the capacity is different in each cell.

5 Acknowledgement

The work of E. CS-V. was supported by NKFIH (National Research, Development, and Innovation Office), Hungary, grant no. K 120558.

References

1. L. Cienciala, L. Ciencialová, and A. Kelemenová. Homogeneous P colonies. *Computing and Informatics*, 27(3):481–496, 2008.
2. L. Ciencialová, L. Cienciala, and P. Sosík. P colonies with evolving environment. In A. Leporati and C. Zandron, editors, *Proceedings of the 17th International Conference on Membrane Computing (CMC17)*, pages 105–118, 2016.
3. L. Ciencialová, E. Csuhaj-Varjú, L. Cienciala, and P. Sosík. P colonies. *Bulletin of the International Membrane Computing Society*, 1(2):119–156, 2016.
4. R. Freund, I. Pérez-Hurtado, A. Riscos-Núñez, and S. Verlan. A formalization of membrane systems with dynamically evolving structures. *International Journal of Computer Mathematics*, 90(4):801–815, 2013.
5. R. Freund and S. Verlan. A formal framework for static (tissue) P systems. In G. Eleftherakis, P. Kefalas, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 8th International Workshop, WMC 2007, Revised Selected and Invited Papers*, volume 4860 of *LNCS*, pages 271–284. Springer, 2007.
6. R. Milner. An algebraic definition of simulation between programs. In *Proceedings of the 2Nd International Joint Conference on Artificial Intelligence*, IJCAI'71, pages 481–489, San Francisco, CA, USA, 1971. Morgan Kaufmann Publishers Inc.
7. G. Păun, G. Rozenberg, and A. Salomaa, editors. *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2009.
8. G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 1–3. Springer, 1997.
9. S. Verlan. Study of language-theoretic computational paradigms inspired by biology. Habilitation thesis, Université Paris Est, 2010.
10. S. Verlan. Using the formal framework for P systems. In A. Alhazov, S. Cojocaru, M. Gheorghe, Y. Rogozhin, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing - 14th International Conference, CMC 2013, Chişinău, Republic of Moldova, August 20-23, 2013, Revised Selected Papers*, volume 8340 of *Lecture Notes in Computer Science*, pages 56–79. Springer, 2013.

A P System for Travelling Salesman Problem

Ping Guo and Yunlei Dai

College of Computer Science, Chongqing University, Chongqing 400044, China

Abstract. P system is a kind of distributed parallel computing model. In the P system, objects in each membrane can follow the evolution of the maximum parallelism principle, so we can solve NP-hard problem in polynomial time. In this paper, we design a family of P system in a uniform way to not only determine whether the travelling salesman problem is solvable but also give the cost and the solution, and then an instance is given to illustrate the feasibility and effectiveness of our designed P systems.

Keywords: P system, Travelling Salesman problem, Membrane Computing, Natural Computing

1 Introduction

Membrane computing is a branch of natural computing which abstract computing models from the architecture and the functioning of living cells, as well as from the organization of cells in tissues, organs (brain included) or other higher order structures such as colonies of cells. The computing models in the framework of membrane computing are usually called P systems, which are distributed and parallel computing models. Many variants of P systems, including cell-like P systems [1, 2], tissue-like P systems [3, 4] and neural-like P systems [5, 6], have been investigated and most of them are proved to be universal and efficient. It has been proved that membrane computing has the same equivalent computing power as Turing machine.

Until now, many kinds of P system have been proposed to solve NP-complete problems, such as SAT [9, 10], HPP [11, 12], arithmetic operations [13, 14] and TSP [15-19]. The TSP (namely travelling salesman problem) is a typical representative of the NP hard problem. To solve the TSP, researchers have proposed many algorithms for decades. According to whether the algorithm is to find the global optimal solution, these algorithms can be divided into two categories: exact algorithms and approximate algorithms. Ref. [15] solves the symmetric travelling salesman problem by using an improved branch and bound algorithm with a new lower bounds, Ref. [16] proposes a novel ant colony optimisation (ACO) algorithm - Moderate Ant System to solve TSP, this algorithm is experimentally turned out to be effective and competitive. In P system, some kinds of P system also have been proposed to solve the TSP. Ref. [17] proposes a heuristic solution to the travelling salesman problem that uses membrane computing to allow for distributed asynchronous parallel computation, and genetic algorithm to select the Hamiltonian cycles that are to be included in the computation in each membrane. However, membranes used in this paper only are structures that hold programs and data, which don′t conform to Gheorghe Păun′ s model of membrane computing. Ref. [18] proposes a new type of approximate algorithms called membrane algorithms for solving travelling salesman problem, a membrane algorithm borrows nested membrane structures and a number of subalgorithms which can be any approximate algorithm for optimization problems are stored in membrane separated regions. Obviously, membrane algorithms don′t conform to Gheorghe Păun′ s model of membrane computing too.

We have already designed a family of P system in a uniform way to not only determine whether there is a Hamiltonian cycle path but also find the complete solution [19]. In this paper, we designed a family of P system in an exact way to solve travelling salesman problem based on the former paper [19]. This P system is constituted by a series of membranes and they are nested one by one. The vertices, edges and weights of edges in an undirected graph will be sent to these membranes. By applying the designed rules to evolve objects, travelling salesman problem can be finally solved. The organization of the rest part of this paper is as follows: section 2 introduces the foundation of P systems, section 3 describes the parallel computing method for solving travelling salesman problem and section 4 proposes our P systems with the multi-membrane structure and rules with priority designed. In section 5, we give an instance to show how to solve travelling salesman problem in our P systems. The conclusions are drawn in the final section.

2 Foundations

Our work in this paper is based on cell-like P systems, so we give the basic concepts about cell-like P systems firstly.

The structure of cell-like P systems is illustrated in Fig.1 [8]. As suggested by Fig.1, the structure is a hierarchically arranged set of membranes which are usually identified by labels from a given set and contained in a distinguished external membrane. If a membrane does not contain any other membrane, it is called elementary. The membrane that contains all the other membranes is referred as the skin. Each membrane determines a region delimited from above by it and from below by the membranes placed directly inside, if any exists.

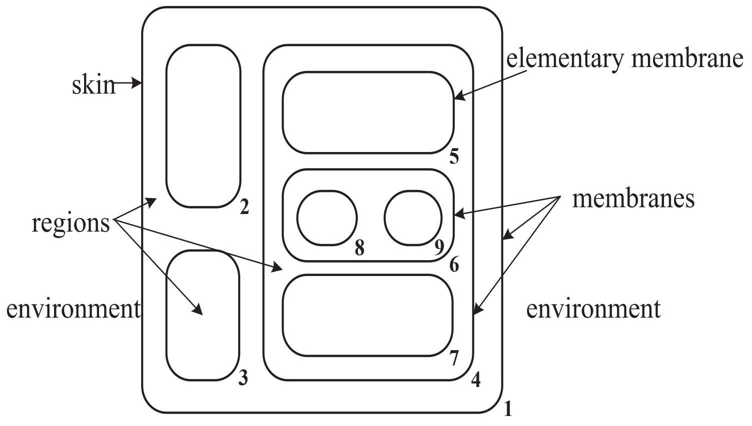


Fig.1. The structure of cell-like P system

Formally, a cell-like P system (of degree $m \geq 1$) can be defined as form[7,8]:

$$\Pi = (O, \mu, \omega_1, \cdots, \omega_m, R_1, \cdots, R_m, i_o) \tag{1}$$

Where,

1. O is the alphabet. Each symbol represents one kind of objects in the systems, O^* is the non-empty Kleene closure over O, where λ is empty string, $O^+ = O^* - \{\lambda\}$;
2. μ is a membrane structure with m membrane, labeled by 1, 2, $\cdots$, m.
3. ω_i $(1 \leq i \leq m)$ is a string over O and represents the multiset of objects placed in membrane i.For example, there are 5 copies of object a and 3 copies of object b in membrane i, then we have $\omega_i = a^5b^3$. $\omega_i = \lambda$ means that there is no object in the membrane i.
4. The rules shall have priority in R, describe as $(u \rightarrow v,\ k)$, where $u \rightarrow v$ is a rewrite rule, and k indicates the priority, the smaller value k is set, the higher the priority of the corresponding rule is. When k =1, the corresponding rule will have the highest priority.
5. R_1, R_2, $\cdots$, R_m are finite sets of possible evolution rules over O associated with the regions 1, 2, $\cdots$, m of μ. The rules in R_i $(1 \leq i \leq m)$ are of the form $U \rightarrow V|_a$, with $a \in O$, $U \in O^+$, $V = V^{'}$ or $V = V^{'}\delta$, $V^{'} \in (O \times Tar)^*$, and $Tar = \{here; out; in_j|1 \leq j \leq m\}$. $Here$ means V is remained in the same region, out means V goes out of the region, and in_j means V goes to inner membrane j.Object a is a promoter in rule $U \rightarrow V|_a$, this rule can only be applied in the presence of object a.
6. i_o is output region of the system and it saves the final result.

In each membrane, rules are applied according to the following principles:

1. Non-determinism. Suppose n rules compete for the reactants which can only support m $(m < n)$ rules to be applied, then the m rules are chosen non-deterministically.
2. Maximal parallelism. All of the rules that can be applied must be applied simultaneously.

3 Travelling Salesman Problem and the parallel algorithm

3.1 The algorithm for Travelling Salesman Problem in $\Pi_{\mathbf{TSP}}$

Travelling salesman problem is a well-known NP-complete problem in graph theory. Actually, travelling salesman problem has been widely used in transportation, computer network, circuit board design, logistics distribution and so on. Consequently, research on the parallel computing methods for travelling salesman problem has great significance. Because objects in cell-like P systems can be evolved in accordance with the principle of maximal parallelism, the characteristics of its high parallelism makes it possible to solve NP-complete problems in polynomial time. In this section, we firstly give the definition of travelling salesman problem, then give an algorithm which is suitable for solving travelling salesman problem in P systems.

Given a directed graph G on d vertices, in which each edge (x, y) has a weight $w(x, y)$ associated with it, where x and y are vertices of G. Given a path P in G, the cost $\mathrm{C}(P)$ of path P is the sum of the weights of all edges in P. A path that passes through each vertex of G exactly once and returns to the starting vertex is called a Hamiltonian cycle. The travelling salesman problem looks for a Hamiltonian cycle with minimum cost among all Hamiltonian cycles of G. Fig.2 gives an instance of undirected weighted graph, let V_1 be the starting vertex and ending vertex, its Hamiltonian cycles are $\{\mathrm{V}_1 \rightarrow \mathrm{V}_2 \rightarrow \mathrm{V}_3 \rightarrow \mathrm{V}_4 \rightarrow \mathrm{V}_5 \rightarrow \mathrm{V}_1\}$, $\{\mathrm{V}_1 \rightarrow \mathrm{V}_2 \rightarrow \mathrm{V}_3 \rightarrow \mathrm{V}_5 \rightarrow \mathrm{V}_4 \rightarrow \mathrm{V}_1\}$, $\{\mathrm{V}_1 \rightarrow \mathrm{V}_2 \rightarrow \mathrm{V}_5 \rightarrow \mathrm{V}_3 \rightarrow \mathrm{V}_4 \rightarrow \mathrm{V}_1\}$ and so on. As we can see from Fig.2, the minimum cost cycle is the second Hamiltonian cycle, so the solution of travelling salesman problem for Fig.2 is $\{\mathrm{V}_1 \rightarrow \mathrm{V}_2 \rightarrow \mathrm{V}_3 \rightarrow \mathrm{V}_5 \rightarrow \mathrm{V}_4 \rightarrow \mathrm{V}_1\}$.

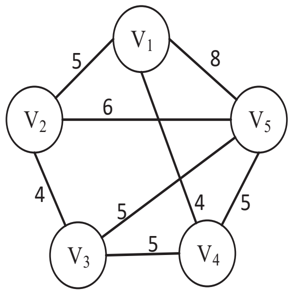


Fig.2. graph G

3.2 The algorithm for Travelling Salesman Problem in $\Pi_{\mathbf{TSP}}$

This section presents the algorithm PATSP to solve travelling salesman problem which is based on the algorighm PAHPP proposed in Ref. [19]. Algorithm PAHPP gives all solutions when Hamiltonian cycle problem is solvable (namely all-HCP). The structure of these solutions construct a multi-tree, the node in the multi-tree are the vertices in the graph. Particularly, the root node is starting vertex. A path starting from the root node to a leaf node is a Hamiltonian cycle path. The PATSP (Parallel algorithm for Hamiltonian cycle problem) can be described as follows:

Table 1. Algorithm: PATSP

```
Input: undirected weighted graph G=(V, E)
Output: the minimum cost cycle path or No
(1)Path construction: construct all legal paths in parallel, all paths make up a multi-tree,
the steps of constructing one legal path P as follows:
   1) Add the starting vertex to the path P as the common root node.
   2) If there is an edge from the last vertex on path P to a vertex has not been added,
add the vertex to path P as a child node.
   3) Repeat step 2) until no vertex could be added to path P.
   4) If each vertex of graph G has been added to path P and there is an edge from the
   last vertex on path P to the starting vertex, add the starting vertex to path P as
   a child node.
(2)Path detection: delete illegal Hamiltonian cycle paths while constructing the paths.
   1) If there is no edge from the last vertex on path P to a vertex has not been added,
   delete path P.
   2) If each vertex of graph G has been added to path P and there is no edge from the
   last vertex on path P to the starting vertex, delete path P.
(3)Path comparison: find a Hamiltonian cycle with minimum cost among all Hamiltonian
cycles of G.
   1) Starting from every leaf node to find the cost of every Hamiltonian cycle path.
   2) If several paths share the common parent node, compare the cost of each path,
   find the path with minimum cost among them.
   3) repeat 2) until the root node has been visited.
(4) Path cutting: delete paths that don′t have the minimum weight.
(5) Output: output travelling salesman path or No.
End
```

4 The design of P Systems $\Pi_{\mathbf{TSP}}$

In this section, we design a P system Π_{TSP} for solving travelling salesman problem based on the algorithm which discussed in subsection 3.2

4.1 The Definition of $\Pi_{\mathbf{TSP}}$

According to the previous definition of eq. (1), the like-cell P systems Π_{TSP} can be defined as:

$$\Pi = (O, \mu, \omega, R, \rho, i_o) \tag{2}$$

where

1. O is a finite and non-empty alphabet of objects, it includes:
   • objects which represent vertices in the undirected weighted graph: $\{a_i, e_i, u_i, p_i, q_i, f_i \mid 1 \leq i \leq \text{n}\}$
   $\{a_1, a_2, \cdots, a_n\}$ represents the set of vertices of graph, simultaneously, $\{e_1, e_2, \cdots, e_n\}$, $\{q_1, q_2, \cdots, q_n\}$, $\{f_1, f_2, \cdots, f_n\}$ also represents the set of vertices of graph. e_1, e_2, u_i, q_i and f_i represent the i-th vertex in the graph ($1 \leq i \leq$n).
   • some special objects:
   -y, w, z: the existence of y means that all vertices have been visited; w, v represents that Hamiltonian path has been found.
   -λ: means that there is no object in the membrane
   -τ: when τ is in membrane i, the thickness of membrane i will be increased, objects can't get in and out of membrane i.
   -δ: membrane i is in membrane j, when object δ is created in membrane i, membrane i will be dissolved and all objects in membrane j will be sent to membrane j.
   Besides the objects mentioned above, there are other objects which are used to control the application of the rules, and we do not interpret them here.
2. μ is the initial structure of our P system as shown in Fig.3, and the structure will be changed during the evolution of the system.

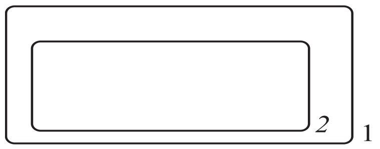


Fig.3. The initial structure of the P system Π_{TSP}

3. ω=$\{\omega_0\}$ is a collection of multisets in the initial configuration, where:ω_0=$\{s^j \mid j = n\text{-}2\} \cup \{p_{io}, b, m, , f_{io}\} \cup \{a_i \mid 1 \leq i \leq \text{n}, i \neq i_o\}$, where n is the number of nodes in the graph, p_{io} means the output path will start from the node i_o ($1 \leq i_o \leq$n), a_i represent the vertices of graph, m and ζ are used to control the application of the rules.
4. The form of rules in R are given in Section 2, and $R = R^{\text{C}} \cup R^{\text{D}} \cup R^{\text{F}} \cup R^{\text{T}}$, where, R^{C} for path construction, R^{D} for path detection, R^{F} for path comparison and R^{T} for path cutting. Based on the Parallel algorithm PATSP, the procedure of applying the rules in Π_{TSP} is:
   - path construction(see subsection 4.2.1)
   - path detection (see subsection 4.2.2)
   - path comparison (see subsection 4.2.3)
   - path cutting (see subsection 4.2.4)

5. the final result can be found in membrane 1 when the whole system halts, i_0 corresponds to membrane 1 in Π_{TSP}.

4.2 The rules in $\Pi_{\mathbf{TSP}}$

Path construction When P system starts, Objects in skin membrane represent the undirected weighted graph: 1) a_i, represent the vertices of graph G; 2) e, represent the end of inputting vertices; 3) f_i, represent the starting and ending vertex of the Hamiltonian cycle.

1. visit each vertex of graph
   If we want to solve travelling salesman problem, we first need to find all Hamiltonian cycle paths. So if we want to search for the Hamiltonian cycle path, we need to visit all nodes exactly once from the starting vertex, and finally return to the starting vertex. Firstly, the length of path P is 0, the length will be increased by 1 when a vertex has been visited. Rules in R^{C} associated with the process are (1≤i≤n, 1≤j≤n, 1≤k≤n):
   r$_1$: $([ba_i]_k \to [e_i c[tp_i]_{k+1}]_k,\ 1)$
   r$_2$: $(u_i \to u_i(a_i,\ in)|_c,\ 2)$
   r$_3$: $(a_i \to a_i(a_i,\ in)|_c,\ 2)$
   r$_4$: $(s \to s(s,\ in)|_c,\ 2)$
   r$_5$: $(f_i \to f_i(f_i,\ in)|_c,\ 2)$
   r$_6$: $(m \to m(m,\ in)|_c,\ 2)$
   r$_7$: $(r \to r(r,\ in)|_c,\ 2)$
   r$_8$: $(\zeta \to \zeta(\zeta,\ in)|_c,\ 2)$
   r$_9$: $(cp_i \to p_i b(q_i\tau, in),\ 3\ )$
   r$_{10}$: $(e_i \to u_i\ |_c,\ 2\ )$
   r$_{11}$: $(ts \to \lambda,\ 2\ )$
   r$_{12}$: $(q_i p_j \to p_j b r^n,\ 1\ )$
   r$_{13}$: $(q_i p_j \to d,\ 1\ )$
   We use rules of type r_1 to create sub-membrane and the sub-membrane is used to determine whether there is an edge between two specified vertices, rules of types $r_2 \sim r_8$ is used to copy and move objects to the new membrane, when rule of type r_{12} is applied, it means that there is an edge from the vertex V$_i$ to the vertex V$_j$ and the weight of edge is n, when rule of type r_{13} is applied, it means that there is no edge from the vertex V$_i$ to the vertex V$_j$. In the process of path construction, after rule of type r_1 is applied, a sub-membrane will be created, then rules of type $r_2 \sim r_8$ will be used to copy and move objects to the new sub-membrane. let V$_i$ be the last vertex of current path, if there is an edge from V$_i$ to V$_j$, rule of type r_{12} will be applied to create object b and create k object r (the number of object r represents the weight of corresponding edge), this means that V$_j$ has been added to the current path; if there is no edge from V$_i$ to V$_j$, then rule of type r_{13} will be applied to create object d, then the new sub-membrane and objects in it will be dissolved, this means that the current path can′t be a Hamiltonian cycle path.
2. Return to the starting vertex
   When each vertex of the graph has been added to the path P, let V$_i$ be the last vertex of path P, we need to determine whether there is an edge from V$_i$ to the starting vertex. If so, path P is a Hamiltonian cycle path; if not, path P is not a Hamiltonian cycle path. Rules in R^{C} are as follows (1≤i≤n):
   r$_{14}$: $(bf_i \to y[p_i],\ 3)$
   r$_{15}$: $(yb \to (o; w,\ out),\ 1)$
   r$_{16}$: $(yd \to d\delta,\ 1\ )$
   r$_{17}$: $(p_i y \to p_i(yq_i,\ in),\ 1\ )$
   r$_{18}$: $(rp_i \to p_i(r_i,\ out)|_o,\ 1\ )$
   After the usage of rule of type of r_{14}, object p_i which represents the starting vertex and a new sub-membrane will be created. Then after the usage of

rule of type r_{17}, object q_i which represents the last vertex of path P will be created and sent to the sub-membrane. By this time, rule of type r_{12} will be applied to create object b and object r if there is an edge from the last vertex of path P to the starting vertex, then because of the existence of object b, rule r_{15} will be applied to create object o and send object w which shows there exists a Hamiltonian cycle path to outer membrane, and because of the existence of object o, all object r will be converted to r_i to outer membrane. If there is no edge from the last vertex of path P to the starting vertex, rule of type r_{13} will be applied to create object d if there is no edge from the last vertex of path P to the starting vertex, then because of the existence of object d, rule r_{16} will be applied to dissolve the sub-membrane which shows that path P cannot be a Hamiltonian cycle path.

Path detection

1. Judgment
   When rule r_{15} in R^{C} is applied, object w will be created and be sent to outer membrane which means that a Hamiltonian cycle path has been found. Rules in R^{D} associated with the process are:
   r_{19}: $(w \rightarrow z\delta, 1)$ r_{23}: $(vzh \rightarrow v(t,out), 4)$
   r_{20}: $(tz \rightarrow v(t, out), 3)$ r_{24}: $(szh \rightarrow z, 2)$
   r_{21}: $(szt \rightarrow vz, 1)$ r_{25}: $(zh \rightarrow k, 5)$
   r_{22}: $(k \rightarrow h\delta, 1)$
   Rule r_{19} is used to reduce the thickness of membrane and convert object w to object z. The existence of object s means that sub-membranes of current membrane hasn′t been all disposed, so when object s doesn′t exist in current membrane, rule r_{20} will be used to create v and send object t to outer membrane. In Π_{TSP}, the depth of membrane shows the number of vertices which the current path contains, object p_i in a certain membrane represents vertex V_i is on the current path and the number of object v in membrane represents the number of Hamiltonian cycle paths that start from the current path. If object s exists in current membrane, then rule r_{21} will be applied to create object v only.
   If no Hamiltonian path has been found, rule r_{22} will be applied to send object h to outer membranes(the existence of object h exactly means that no Hamiltonian path has been found). If object s exists in outer membrane, rule r_{24} will be used; if only object v exists in outer membrane, that is to say all sub membranes have been disposed and Hamiltonian cycle path exists. At this time, rule r_{23} will be applied to delete object h and to send t to outer membrane; if outer membrane does not exist object s and object v, this means that all sub membranes has been disposed and no Hamiltonian path has been found, then rule r_{25} will be applied to create object k to start delete rules.
2. Pruning
   Pruning is to delete surplus membranes and objects, only membranes and objects which represent Hamiltonian cycle path are remained. In Π_{TSP} , pruning is needed in following four situations:

(a) Visiting each vertex of graph. Let V_i be a vertex which has not been visited. We need to determine whether there is an edge from the last vertex on path P to V_i. In Π_{TSP}, we dispose it in a new sub-membrane which is created by applying rule of type r_1 in R^{C}. When there is no edge from the last vertex on path P to V_i, object d will be created and delete rules in R^{D} will be started. As a result, the new sub-membrane and objects in it will be dissolved. Rules in R^{D} associated with the process are ($1 \leq i \leq$n):
r_{26}: $(s \to \lambda \,|_d, 1)$ r_{31}: $(d \to k\delta, 2)$
r_{27}: $(t \to \lambda \,|_d, 1)$ r_{32}: $(p_i \to \lambda \,|_k, 1)$
r_{28}: $(a_i \to \lambda \,|_d, 1)$ r_{33}: $(\zeta \to \lambda \,|_k, 1)$
r_{29}: $(u_i \to \lambda \,|_d, 1)$ r_{34}: $(m \to \lambda \,|_k, 1)$
r_{30}: $(f_i \to \lambda \,|_d, 1)$ r_{35}: $(r \to \lambda \,|_d, 1)$

(b) All sub membranes have been created. When all sub membranes of current membrane have been created, objects in current membrane except s, p_i should be deleted. Moreover, the thickness of current membrane should be reduced. Rules in R^{D} associated with this process are ($1 \leq i \leq$n):
r_{36}: $(bu_i \to g, 2)$ r_{39}: $(f_i \to \lambda \,|_g, 1)$
r_{37}: $(a_i \to \lambda \,|_g, 1)$ r_{40}: $(g \to z\delta, 2)$
r_{38}: $(u_i \to \lambda \,|_g, 1)$
Object b is used to control the creation of new membrane and object u_i ($1 \leq i \leq$n) represents vertex v_i ($1 \leq i \leq$n) is already added to the path. When object a_i ($1 \leq i \leq$n) doesn′ t exist in current membrane (namely all vertices have been visited),r_{36} which has a lower priority than r_1 in R^{C}, will be applied to generate object g to start rules to delete corresponding objects and membrane.

(c) Each vertex of graph has been added to path P. We need to determine whether the staring vertex could be added to path P when each vertex of graph has been added. If there is no edge to the starting vertex, object d will be created by applying rule r_{13} in R^{C}. Then by the action of rule r_{31} and rule r_{22} in R^{D}, the sub-membrane and objects in it will be dissolved, this means that path P is not a Hamiltonian cycle path. Rules in R^{D} associated with this process are ($1 \leq i \leq$n):
r_{31}: $(d \to k\delta, 2)$ r_{22}: $(k \to h\delta, 1)$
r_{32}: $(p_i \to \lambda \,|_k, 1)$

(d) All sub membranes have been detected and no Hamiltonian path has been found in these sub membranes, In this situation, the current membrane should be dissolved. Rules in R^{D} associated with this process are ($1 \leq i \leq$n):
r_{32}: $(p_i \to \lambda \,|_k, 1)$ r_{25}: $(zh \to k, 5\ )$
r_{22}: $(k \to h\delta, 1)$

Path comparison When all Hamiltonian cycle paths have been constructed, we need to find a path with a minimum cost among all Hamiltonian cycle paths. Starting from the innermost membrane to skin membrane, we move object r_i(1≤i≤n) whose number represent the cost of one Hamiltonian cycle path to outer membranes and compare different paths to find a path with a minimum cost. Rules in R^{F} associated with this process (1≤i≤n, 1≤j≤n):

r$_{41}$: $(r_i m \to c_i r_i \alpha_i, 1)$	r$_{46}$: $(\beta_i \to \lambda\ |_{yi}, 1)$
r$_{42}$: $(r_i \to \beta_i|_{ci}, 1)$	r$_{47}$: $(\alpha_i \to \lambda\ |_{yi}, 1)$
r$_{43}$: $(c_i v \to m|_{\beta i}, 1)$	r$_{48}$: $(\gamma \to \lambda|_{yi}, 1)$
r$_{44}$: $(\beta_i \beta_j \to \beta\gamma, 2)$	r$_{49}$: $(\beta\zeta\alpha_i \to \beta\zeta_i, 2)$
r$_{45}$: $(\beta_i\gamma \to y_i, 1)$	r$_{50}$: $(\beta \to \beta_i\ |_{\zeta i}, 1)$

The strategy of our comparison is pairwise comparison, rule of type r_{41} and r_{42} is used to control that only two Hamiltonian cycle paths are compared everytime. Because of the uniqueness of object m in a membrane, object r_i will be converted to object β_i sequentially. The number of object r_i and r_j represents the cost of two different Hamiltonian cycle paths (path i and path j), the subscript of object r is decided by the subscript of object p in the corresponding sub-membrane. When object r_i and r_j has been converted to β_i and β_j by applying rule r_{42}, rule of type r_{44} will be used to convert β_i and β_j to β. Assume that the number of β_i is less than β_j, which means that the cost of path i is smaller than path j. So after rule r_{44} is applied, β_j will be left. Then rule of type r_{45}, r_{46} and r_{47} will be applied to delete α_j and all of object β_j. What s more, rule of type r_{48} will be applied to delete object γ.

After all object r_j has been deleted, we need to convert β to β_i for letting the process of comparison continue. When rule of type r_{49} is applied, α_i will be dissolved and ζ will be converted to ζ_i . And because the existence of object ζ_i, β will be converted to object β_i under the application of rule r_{50}. By now, a pairwise comparison has completed, object r_j which represent the larger cost of a Hamiltonian cycle path has been all deleted. Rules in R^{P} will applied until all object r_k (1≤k≤n) which don′ t represent the Hamiltonian cycle path a minimum cost in the membrane are deleted.

Path cutting When a Hamiltonian cycle path has been detected that doesn′ t have a minimum cost, we need to delete corresponding membranes that represent this Hamiltonian cycle path. Rules in R^{T} associated with the process are (1≤i≤n, 1≤j≤n):

r$_{51}$: $(y_i \to (y_i, \text{in})|_{\neg\alpha i}, 1)$	r$_{54}$: $(xp_i \to d(x, \text{in}), 1)$
r$_{52}$: $(y_i p_i \to p_i x, 1)$	r$_{55}$: $(\beta_i \to n_i\ |_{\neg v}, 1)$
r$_{53}$: $(y_i p_j \to (y_i, \text{out}), 1)$	r$_{56}$: $(n_i \to (r_j, \text{out})|_{pj}, 1)$

Object y is used to start delete rules, the subscript of object y is decided by the subscript of object p in the corresponding sub-membrane. When object y_i exists and α_i is dissolved, a Hamiltonian cycle path has been detected that doesn′ t have a minimum cost. Then under the application of rule r_{51}, r_{52} and r_{53}, object y_i will get in and out sub-membranes continuously until the subscript of object p in a membrane is the same as y_i. After rule r_{52} in sub-membrane is applied, object x will be created. When object x exists, rule r_{54} will be applied to create object d and send object x into sub-membrane. Because of the existence of d, the membrane and objects in it will be dissolved. With the implementation of rule r_{54}, all corresponding sub-membranes will be dissolved. Once object v doesn$'$ t exist in membrane, path comparison in this membrane has completed, we need to move object r_i to outer membrane. Rule r_{55} will convert β_i to n_i when object v doesn't exist in membrane, then all object n_i will be convert into r_j and be sent out because of the existence of object p_j.

4.3 Parallelism analysis of $\mathbf{\Pi_{TSP}}$

Analysis of $\mathbf{\Pi_{TSP}}$ For a complete undirected weighted graph with n vertices, we can see that the number of all possible Hamiltonian cycle path is at most n! by using the exhaustive method. So as long as taking n! case into account and judging that whether each case can constitute a ring, we can find all the Hamilton loop. As is shown in Fig.4, this process could be described as construct a multi-tree where each possible Hamiltonian cycle path could be found. When the P system starts, let the starting vertex be V_1, in the outermost membrane is the objects represent the rest vertices $V_2 \sim V_n$. For every V_i (2$\leq i \leq$n) hasn$'$ t been visited, we create a new membrane to judge whether there is an edge from V_1 to V_i (corresponding rule is r_1). If the two vertices are connected, the new sub-membranes will be remained(corresponding rules are $r_2 \sim r_8$). Then for the n-2 vertices that haven$'$ t been visited, n-2 new membranes will be created continuously in just sub-membranes. The process will be repeated until each vertex is on the path. What can be summarized by the above process is n! case could all be taken into consideration. For each of the generated path, we judge whether there is an edge between the two vertices in the new sub-membrane(corresponding rules are r_{12} and r_{13}). If there is an edge connected between two vertices, then the sub- membrane will be remained; otherwise, we need to dissolve surplus membranes and pruning is needed in four situations in Π_{TSP}(crucial corresponding rules are r_{13}, r_{22}, r_{31} and r_{40}). So when the process of path detection is completed, only membranes that represents Hamiltonian cycle paths will be remained.

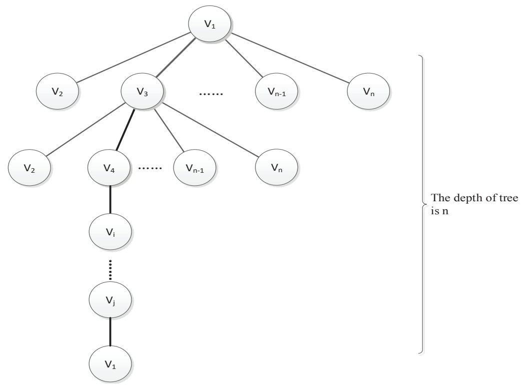


Fig.4. The process of constructing a multi-tree

Hamiltonian cycle paths are represented by a series of membranes that are nested one by one in our P system. As described in algorithm PATSP, all Hamiltonian cycle paths constitutes a multi-tree together. Because each leaf node represents a Hamiltonian cycle paths, so to find the solution of travelling salesman problem, we only need start from the leave nodes of the multi-tree to compare the weight of each Hamiltonian cycle path until we find the Hamiltonian cycle path with minimum weight. In our P system, starting from the innermost membranes, then compare the weight of each Hamiltonian cycle path(corresponding rules are $r_{41} \sim r_{44}$) and delete the the corresponding membrane structures(corresponding rules are $r_{45} \sim r_{48}$ and $r_{51} \sim r_{54}$) represents Hamiltonian cycle paths without a minimum weight. What$'$ s more, transfer the weight of the path has a bigger weight to outer membrane(corresponding rules are $r_{55} \sim r_{56}$) and continue the process of comparison until we find the path with a global minimum weight. What can be summarized by the above process is the right result will be generated when the whole system halts.

Analysis of time complexity According to the maximum parallelism of P systems, the rules that meet their requirements will be executed at the same time. As shown in Fig.5, it is the process of the execution of rules in Π_{TSP}. We assume that the time cost for executing a rule is a slice. What$'$ s more, rules like $[r_i]$ means r_i could be executed or not in Fig.5.

1. Cost of path construction
   The process of path construction is to use the parallel computing methods to construct every possible Hamiltonian cycle path. For a complete undirected weighted graph with n vertices, the number of vertex on a Hamiltonian cycle path is n too. For the i^{th} vertex on the path, there are n-i vertices that should be taken into consideration(n-i+4 slices). So the whole process will take $\sum_{i=1}^{n}(n-i+4)$ slices.
2. Cost of path detection
   The process of path detection and path construction happens at the same time. Dissolving membranes that represent one illegal Hamiltonian cycle path cost constant time(up to 3 slices). Because path construction and path detection happens parallel, so it cost about 3*n slices in total.
3. Cost of path comparison
   One comparison costs 8 slices. Because the process of path comparison is parallel and starts from the innermost membranes. For an undirected weighted graph with n vertices, the depth of membranes is n. So it cost 8n slices in total.
4. Cost of path cutting
   The process of path cutting happens at the same time with path comparison. One path cutting costs 3 slices. When the depth of membranes is n, the total cost of path cutting is 3n slices. In summary, the total cost of Π_{TSP} can be computed as follows: T_{TSP}=$\sum_{i=1}^{n}(n-i+4)$+3*n+8*n+3*n=$\frac{1}{2}$n(n-1)+18n=O(n^2).

Ref. [17] uses RanGen (Randomly Generating) MCGA (Membrane-Computing-Genetic-Algorithm) to solve travelling salesman problem, the time complexity of the algorithm is $O(n^3)$ time. This computation is much faster than that of brute force complete enumeration method in serial, but is still slower than PATSP algorithm. Compared with the traditional ant colony algorithm and genetic algorithm, our algorithm is not only better in time complexity, but also can solve the exact solution of the problem.

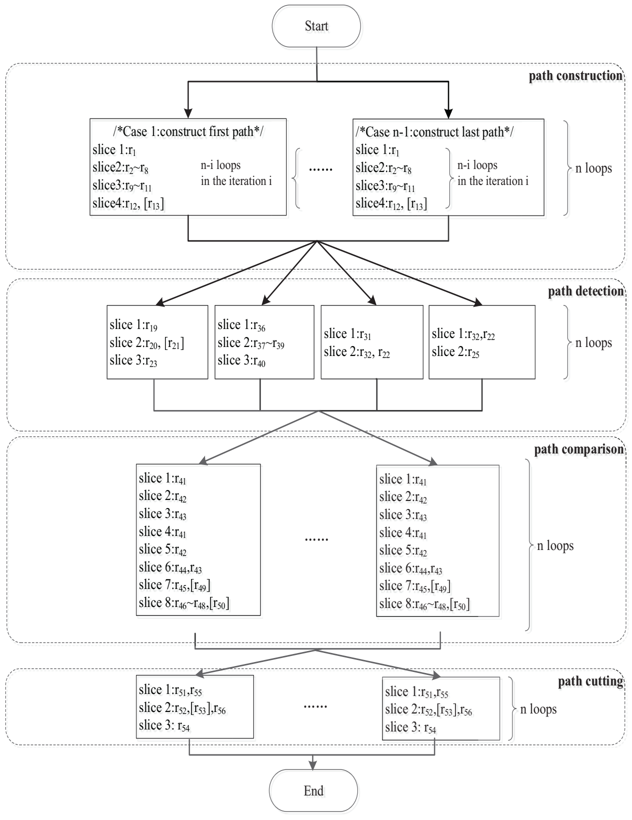


Fig.5. the process of execution of the rules in Π_{TSP}

5 Calculate instance

In this section, we will give an instance to show how to solve travelling salesman problem in P systems Π_{TSP}.

The undirected weighted graph G=(V, E) is shown in Fig.2, let V_1 be the starting and ending vertex, the rules in P systems can be applied as follows:

1. Path construction Objects represent the undirected weighted graph should be input to the skin membrane. Firstly, input multiset $p_1a_2a_3a_4a_5$, then input f_1, last input $mbs_3\zeta$. We will construct legal paths by membrane creation. The available rules in R^{C} are applied in the order of $\{r_1\}\rightarrow\{r_2\sim r_8\}$

$\rightarrow\{r_9 \sim r_{11}\}\rightarrow\{r_{12}\}$. Membrane 2 has the multiset $s^3p_1ba_2a_3a_4a_5f_1m\zeta$, rule of type r_1 is used to create sub-membrane, rules of type $r_2 \sim r_8$ are applied to copy objects to new sub-membrane. Firstly, the length of current path is 1, object p_1 shows that V_1 as the starting vertex has been added to path. We need to determine whether there is an edge from V_1 to V_2, V_3, V_4 or V_5. After rule of type r_1 and rules of type $r_2 \sim r_8$ in R^{C} are applied, a sub-membrane will be created and have multiset $s^3tq_1p_2a_3a_4a_5f_1m\tau\zeta$. As shown in Fig.2, there is an edge from V_1 to V_2. So rule of type r_{12} in R^{C} will be applied, multiset q_1p_2 are converted to p_2br^5 and the new sub-membrane will not be dissolved, this means V_2 has been added to current path. Objects in membrane 2 continue to evolve and this process is corresponding to determine whether there is an edge from V_1 to V_3, V_4 or V_5. After rules of type $r_2 \sim r_8$ in R^{C} are applied in the new sub-membrane mentioned above, it has multiset $s^2p_2ba_3a_4a_5f_1\tau$ which shows that V_2 is the last vertex on current path and V_3, V_4 and V_5 haven't been visited. Above described process of extending the current path can cycle in the new sub-membrane.
When each vertex of graph has been added to current path P, we need to determine whether there is an edge from the last vertex of current path to the starting vertex. Let V_5 be the last vertex on path P, as shown in Fig.2, there is an edge from V_5 to V_1, so after rule of type r_{14} and r_{17} in R^{C} is applied, rule r_{12} will be applied which means V_1 could be added to path P. So path P is a Hamiltonian cycle path.

2. Path detection
   (a) Judgment
   As shown in Fig.6, after rule r_{15} in R^{C} are applied, object w will be created and sent to membrane 5. The existence of object w means that there exists a Hamiltonian cycle path. At this time, membrane 5 has multiset $wtp_5m\tau\zeta r^{27}$. Because of object exists, so after rule r_{19} in R^{D} is applied, membrane 5 will have multiset $mtp_5z\zeta r^{27}$. Then rule r_{20} in R^{D} applied to create object v and send object t to membrane 4. Then rule r_{21} will be applied to evolve multiset szt to vz.

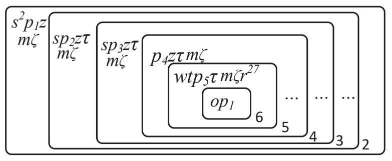


Fig.6. exist a Hamiltonian cycle path

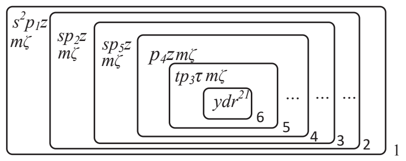


Fig.7. doesnt exist a Hamiltonian cycle path

As shown in Fig.7, when object d is created in membrane 6, it means that there exists no Hamiltonian cycle path. At this time, Rule r_{16} will be applied to dissolve membrane 6 and object d will be sent to membrane 5. Because the existence of object d, object t in membrane 5 will be dissolved. Then rule r_{31} in R^{D} will be applied to reduce the thickness of membrane 5 and convert object d to object k. Because of the existence of object k, rule r_{32} in R^{D} will be applied to dissolve object p_3 in membrane 5.

(b) Pruning

Surplus membranes and objects will be dissolved, only membranes and objects that represent Hamiltonian cycle path are remained. Relevant rules are applied in this instance as follows:

①When visiting each vertex of graph, let v_i be a vertex which we have not visited. We need to determine whether there is an edge from the last vertex on path P to v_i. As shown in Fig.8, rule of type r_{13} will be applied to create object d because there is no edge from V_1 to V_3. Then rules in R^{D} will be applied to dissolve membrane 3 and all objects in it.

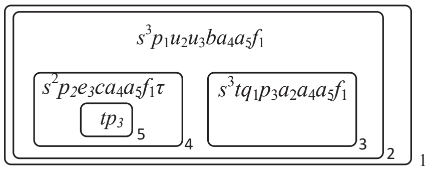


Fig.8. visit vertex of graph G

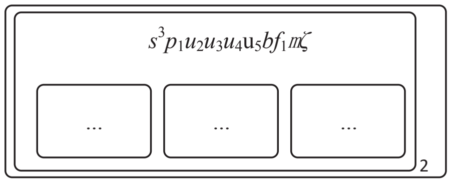


Fig.9. all object a_i has evolved to u_i in membrane 2

②As shown in Fig.9, all object a_i has evolved to u_i which means that we have tried adding each vertex to current path. Then we need to dissolve objects in membrane 2 except s, p_i. Rules in R^{D} are applied in the order of $\{r_{36}\}\rightarrow\{r_{37}, r_{38}, r_{39}\}\rightarrow\{r_{40}\}$.

③As shown in Fig.10, each vertex has been added to current path. But because there is no edge from V_3 to V_1, rule of type r_{13} in R^{C} is applied to dissolve membrane 6 and object d will be sent into membrane 5, then object d will evolve to object k under the action of rule r_{31} in R^{D}.

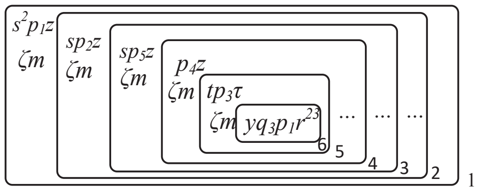


Fig.10. each vertex has been added to current path

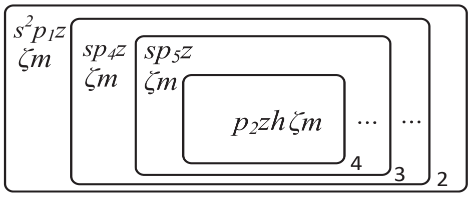


Fig.11. all sub membranes of membrane 4 have been disposed

④As shown in Fig.11, all sub membranes of membrane 4 have been disposed and there exists no Hamiltonian cycle path. Membrane 4 and object p_2 in it will be dissolved by the action of rule r_{25} and rule of type r_{32} in R^{D}. After rules in R^{D} are applied, object h which shows there exists no Hamiltonian cycle path will be sent into membrane 3.

3. Path comparison

As shown in Fig.12, two Hamiltonian cycle paths has been found in membrane with the cost of 23 and 27. We need to find the smaller one between the cost of two Hamiltonian cycle paths. Because of using of rule r_{18} in R^{C}, all object r will be sent out from the innermost membrane. After rule r_{19} and r_{20} in R^{D} is applied in membrane, the number of object v in membrane 5 is 1, after rule of type r_{41} and r_{42} in R^{F} is applied, all object r_1 has been converted to object β_1. Then rule of type r_{55} and r_{56} will be applied to convert object β_1 to object r_5 and send all object r_5 to membrane 4. Similar to the application of rules in membrane 5, all object r_5 will be converted to object r_4 and will be sent into membrane 3. By now, the membrane structure is shown in Fig.13.

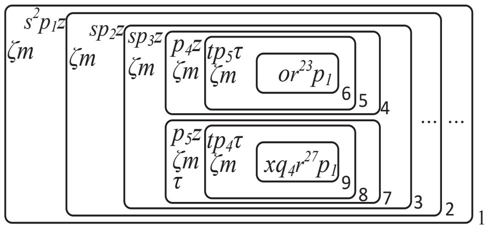


Fig.12. two Hamiltonian cycle paths

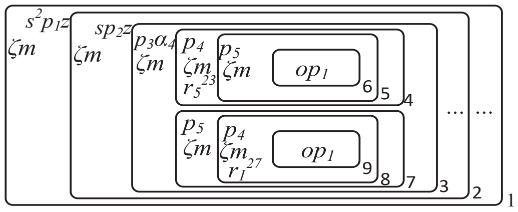


Fig.13. the membrane structure in path comparison

Rule of type r_{41} in R^{F} is used to create object c_4 which is used to convert all object r_4 to β_4 and because of the existence of object β_4, object c_4 will be converted to object m. As is shown in Fig.14, all object r_4 in membrane 7 will also be sent into membrane 3 and will be converted to object r_5. After rule of type $r_{41}{\sim}r_{43}$ in R^{F} is used, object r_5 will be converted to object β_5, then the comparison of the cost of two Hamiltonian cycle path will start. After rule of type rule r_{44} in R^{F} is applied, object β_4 and object β_5 are converted to β. And three object are left in membrane 3. So rule of type r_{45} will be used next to create object y_5 which is used to delete object β_5, α_i, and γ. By now, objects r_5 which represents the larger cost of two Hamiltonian cycle paths has been all deleted. Rule of type r_{49} in R^{F} is used to create object

ζ_4 which is used to convert β to β_4. By now, a path comparison has been completed which is shown in Fig.15.

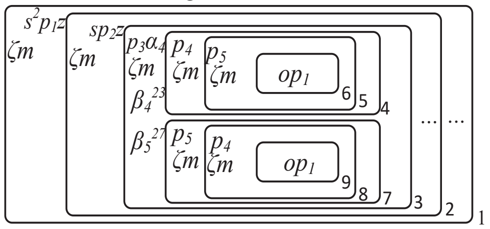


Fig.14. the membrane structure in path comparison

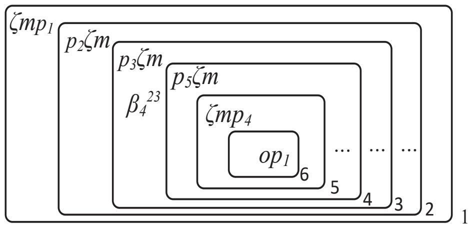


Fig.15. the membrane structure after path comparison

4. Path cutting

After a path comparison, we have known membranes and objects which represent a Hamiltonian cycle path with a larger cost. As shown in Fig.15, membrane 7 and its sub-membranes should be dissolved. Object α_5 has been deleted because it represent the path with a larger cost. By applying the rule of type r_{51}~r_{53} in R^{T}, object y_5 will be sent into a sub-membrane which has object p_5. Then by applying rule of type r_{54} in R^{T} continuously, object d will be created to start delete rules. As a result, corresponding membranes and objects will be dissolved. What$'$ s more, β_4 will be converted to n_4 by applying rule of type r_{55} in R^{T} because object v has been all dissolved which means that a path comparison has been completed. Then n_4 will be converted to object r_3 and will be sent to outer membrane to start a new path comparison because of the existence of object p_3. By now, a path cutting has been completed.

5. Final result

When the whole system halts, the final membrane structure is shown in Fig.16. As we can see in Fig.16, only membranes and objects that represent the Hamiltonian path with a minimum cost are remained. Object p_i represents vertex v_i in gragh. By detect object p_i in each membranes,we knows the path is:$\{\mathrm{V}_1 \rightarrow \mathrm{V}_2 \rightarrow \mathrm{V}_3 \rightarrow \mathrm{V}_5 \rightarrow \mathrm{V}_4 \rightarrow \mathrm{V}_1\}$.

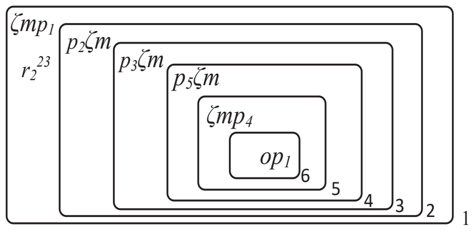


Fig.16. the final membrane structure

6 Conclusions

Membrane computing is a new computing model inspired by biochemical reaction in living cells and transmission of compounds through membranes. This paper presents a family of P systems to solve travelling salesman problem in O(n^2) time. In this P system, we firstly construct all Hamiltonian cycle paths by membrane creation which is based on our former paper, then find the Hamiltonian cycle path with a minimum cost, lastly those membranes and objects which represent Hamiltonian cycle paths that don$'$ t have a minimum cost will be deleted by path cutting. An instance is given to illustrate the feasibility and effectiveness of our designed P system. Improving our algorithm that it can cope with multiple shortest pathes as a TSP solution is our main future research work.

References

1. A.Vitale, G.Mauri, C.Zandron. Simulation of a bounded symport/antiport P system with Brane calculi[J]. Biosystems, 2008, 91(3): 558-571.
2. C.Martin-Vide, Gh. Păun, A. Rodríguez-Patón. On P systems with membrane creation[J]. Computer Science Journal of Moldova, 2001, 9(2): 134-145.
3. C.Mart, Gh. Păun, J.Pazos. Tissue P systems[J]. Theoretical Computer Science, 2003, 296(2): 295-326.
4. R. Freund, Gh. Păun, M. J. Pérez-Jiménez. Tissue P systems with channel states[J]. Theoretical Computer Science, 2005, 330(1):101-116.
5. X. Y. Zhang, X. X. Zeng, B. Luo, and J. B. Xu, J. Comput. Theoret. Nanosci. 9, 769 (2012).
6. T. Song, Y. Jiang, X. L. Shi, and X. X. Zeng, J. Comput. Theoret. Nanosci. 10, 999 (2013).
7. Gh. Păun, Computing with membranes. Journal of Computer and System Sciences, 61(2000): 108C143.
8. Gh. Păun, Membrane Computing. An Introduction, Springer-Verlag, Berln, 2002.
9. P. Guo, J.-F. Ji, H.-Z. Chen, R. Liu, Solving All-SAT Problems by P Systems, Chinese Journal of Electronics, 24(4) (2015) 744-749.
10. P. Guo, J. Zhu, M. Q. Zhou, A family of uniform P systems for All-SAT problem, Journal of Computational and Theoretical Nanoscience, 13(1) (2016) 319-326.
11. L. Pan, A. Alhazov. Solving HPP and SAT by P systems with active membranes and separation rules[J]. Acta Informatica, 2006, 43(2):13 l-145.

12. K Ishii, A. Fujiwara, Asynchronous P systems for SAT and Hamiltonian Cycle Problem, in: 2010 Second World Congress on Nature & Biologically Inspired Computing, IEEE, 2010, pp. 513-519.
13. P. Guo, J. Chen. Arithmetic operation in membrane system. In: Proceedings of International Conference on BioMedical Engineering and Informatics. Sanya, Hainan, China. 2008: 231-234.
14. P. Guo, H. Z. Chen, H. Zheng. Arithmetic Expression Evaluations with Membranes[J]. Chinese Journal of Electronics, 2014, 23(1): 55-60.
15. M. Padberm, G. Rinaldi. A Branch-And-Cut Algorithm For The Resolution Of Large-Scale Symmetric Traveling Salesman Problems. Society for Industrial and Applied Mathematics, Vol. 33, No. 1, pp. 60-100, March 1991
16. P. Guo, Z. J. Liu. Moderate ant system: An improved algorithm for solving TSP[C]. 7^{th} International Conference on Natural Computation, pp. 1190-1196, 2011.
17. P. Manalastas. Membrane Computing with Genetic Algorithm for the Travelling Salesman Problem. Theory and Practice of Computations, (2013) 116-123 .
18. T. Y. Nishida, Membrane Algorithms: Approximate Algorithms for NP-Complete Optimization Problems. Springer Berlin Heidelberg, (2006) 303-314.
19. P. Guo, Y. L. Dai, H. Z. Chen, A P system for Hamiltonian cycle problem, Optik, 127(20) (2016) 8461C8468.

Event-based Life in a Nutshell: How Evaluation of Individual Life Cycles Can Reveal Statistical Inferences using Action-accumulating P Systems

Thomas Hinze[1] and Benjamin Förster[2]

[1]Friedrich Schiller University Jena, Department of Bioinformatics
Ernst-Abbe-Platz 2, D-07743 Jena, Germany

[2]Brandenburg University of Technology, Institute of Computer Science
Postfach 10 13 44, D-03013 Cottbus, Germany

`thomas.hinze@uni-jena.de, benjamin.foerster@b-tu.de`

Abstract. A sequence of perceivable *events* or recorded observations over time commonly witnesses the *life cycle* of an individual at a macroscopic perspective. In case of a human being, birth could make the starting point followed by successive maturation along with increase of individual skills. Further events like foundation of a family, stages of career, coping with dramatic diseases, loss of abilities, and finally the death mark crucial events within a human life cycle. Even beyond biology, life cycles are present in various contexts, for instance when elucidating the quality of durable technical products such as cars. Social scenarios or games with several players incorporate consideration of life cycles as well. Provided by logfiles or monitoring reports, dedicated accumulation of events facilitates identification of life cycles whose statistical analysis promises valuable insights. To this end, we formalise an *individual* by a set of *attributes*. Based on its underlying initial assignment ("genetic potential"), events can *update* corresponding attribute values. Furthermore, events might *create* new individuals but also *kill* or *merge* existing ones. For modelling and evaluation of life cycles, we introduce *action-accumulating P systems* inspired by dealing with events which in turn result in actions at the system's level. Two case studies demonstrate practical benefits from our approach: We explore the survival of pieces in the board game *Mensch ärgere Dich nicht* (Man, don't get annoyed – a variation of Ludo). Secondly, we interpret pseudonymised data from 1,108 students who attended our *university course Introduction to Programming* stating main factors to improve the final grade with emphasis on the effect of passing a line of exercises and practical training offers.

1 Introduction and Background

In the end of 2016, the total amount of electronically stored data worldwide sums up to approximately $8 \cdot 10^{21}$ bytes (8 zettabytes) based on the overall capacity of sold storage media taking into account an average period of usage by five years

[9]. Complementary studies conclude that this pool of data doubles every two up to three years which implies an exponential growth [14]. No doubt, generation and presence of processible data emerged as an essential part of modern life and for sciences in many facets during the last decades. Terms like *data mining* [13], *big data* [6, 7], *knowledge retrieval* [1, 5], or *machine learning* [16, 25] reflect this development. Interestingly, a variety of different *sources* and contexts exists in which data have been produced. On the one hand, *human activities* in the internet, especially in social networks, newsgroups, or online services, originate a plethora of abundant data streams. On the other hand, more and more data come into existence by *measurement*, *technical cognition*, *smart devices*, and *monitoring*. It is said that one minute of autonomous driving induces more than 10 gigabytes [4].

It seems that the tremendous acceleration in throughput is closely connected with shortening the period of time in which new media, new tools, and new techniques for processing of corresponding data become established [23]. For the early mankind, it took more than 100, 000 years to form spoken natural languages for individual communication. Handwriting dates back for nearly 7, 000 years, while printing is available for more than 500 years now. Afterwards, the periods noticeably diminish: Around 120 years ago, the telephone was invented. Mobile phones needed less than 10 years to outperform the conventional telephone in use. The internet gave a further acceleration. First-generation social media platforms reduced its introductory phase to 5 years. Meanwhile, messenger applications have already found widespread practice after two years. What stands out is that each new innovation entails more and more *recorded data* accompanying everybody's life.

From a perspective exclusively focussing on data tracks, a typical human life consists of a sequence of *events* at different points in time in which every event contributes an additional record of data. In the long term, effects initiated by these events can accumulate in an appropriate manner. The underlying processes gradually imply the formation of a personality with an individual setting of skills, capabilities, properties, and qualities. Coinciding with a basic law of dialectics, quantitative change leads to qualitative change: A simple example is given by students attending a university course. During the course, they receive knowledge within the regular lectures. From time to time, taken as data-producing events, they submit solutions to mandatory or optional exercises whose degree of success becomes individually evaluated together with a feedback. After a certain period of training, most of the students possess new or extended abilities related to the topics of the course at various levels of quality, see Figure 1.

In more general terms, the *life cycle* of an *individual* can be abstractly expressed by a temporal sequence of *events* in which each event might slightly modify one or more *attributes*. We assume that every individual is equipped with quantifiable attributes which stand for dedicated skills or qualities. For a majority of events, an update or reset of corresponding attribute values in a specific subset of individuals is sufficient in order to describe the underlying effect. But in some cases, an event affects the existence of individuals under study. To

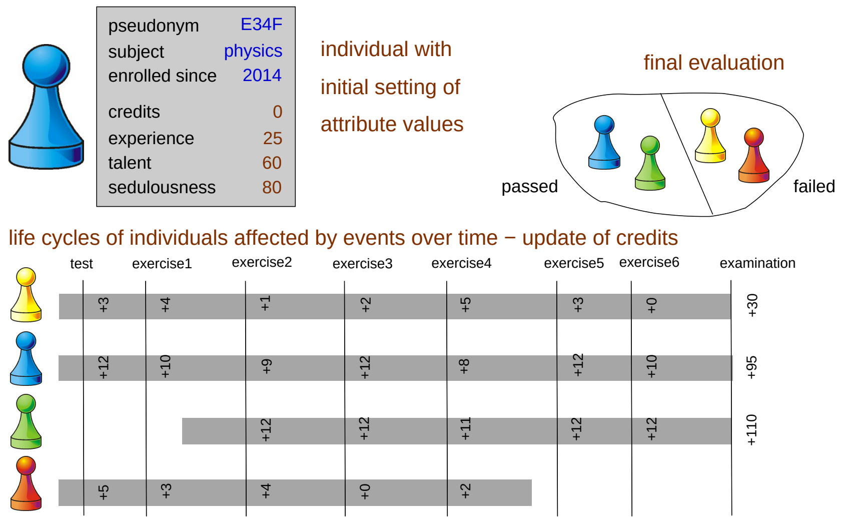


Fig. 1. Students attending a university course constitute individuals attached with attributes such as *subject* or *credits*. Initial attribute values symbolise the personal potential together with classifiable identity information. During the semester, several tests and exercises act as events at certain points in time. Events are able to update some attributes for a selection of students. Some students join the course late, others leave prematurely. Finally, the students can be classified according to an evaluation of their attribute values. The temporal evolution of attribute values opens employment of statistical analysis techniques.

this end, we distinguish several types of events whose effects enlarge or reduce the population of individuals in a dynamical manner. Event-based *creation* of a new individual enables scenarios in which controlled (re)production of individuals over time forms an essential feature. Attribute values of new individuals can be either *cloned* from present ones or alternatively set from scratch. In the opposite sense, an event might result in *killing* a specific subset of individuals from the population. Moreover, it appears to be helpful to have at hand a type of event able to *merge* two previously separate individuals into one individual whose attribute values arise from its precursors in a freely configurable way.

We start with a given multiset of individuals, each of them independently equipped with initial attribute values assuming that all individuals comprise the same composition of attributes. Additionally, we employ a set of events in which each event is characterised by its point in time, its type and necessary parameters and arithmetic functions in order to formalise its effect. A global clock controls progression of time. Processing of events follows the chronological order. Concurrent events are allowed iff they are independent from each other or they are confluent meaning that the final outcome of their effects remain the same. In the course of processing the events, the evolution of individuals and their specific attribute values gets permanently recorded at the granularity of a global clock tick. Based on this cumulative record, final evaluations as a part of the overall system might give new insights into behavioural patterns and their laws. Due to its principle of operation, we name the resulting framework *action-accumulating P systems* aimed at practical exploration of life cycles in various contexts but apart from pure theoretical aspects covered by temporal logics of actions [10, 19] or related formalisms. Instead, our contribution is dedicated to learn more about crucial indicators and indications responsible for clustering [17] or distribution of qualitative factors, their correlations and significance within a dynamical population of individuals. In consequence, this can lead to optimised and efficient behavioural strategies.

Within a large pool of comparable life cycles, statistical methods might allow identification of significant patterns able to substantiate prognoses and inferences. Social and medical sciences frequently practise this idea preferably by questionnaires or behavioural experiments acting in the role of events [8, 22]. Mainly inspired by this concept, our paper presents a way to utilise the framework of P systems in order to capture event-based life cycles for a dynamically managed multiset of attributed individuals together with purposive instruments for evaluation and hypothesis tests. Due to the algebraic nature of all relevant components and due to the fact that clusters of individuals with similar qualities constitute virtual membranes, we have an intuitive relation to the field of membrane computing. To our best knowledge, this is the first attempt to do so. Please note that an individual's membership in a membrane following the classical intention of membrane computing can be managed by a corresponding attribute as well without any loss of information or expressiveness.

Exploration of event-based life cycles is not restricted to studies in systems biology, medicine, and social sciences [12]. Modern technical products like cars or computers exhibit a high degree of inherent complexity. Commonly, they consist of numerous assembled components which in turn have been put together from subcomponents up to the level of elementary parts. Industry and customers are interested in getting detailed statistical information about durability of the entire product and about the resilience of components and parts on their own. As a result, the failure rate of a technical product over time can be obtained based on a representative sample [21]. Typically, the corresponding curve resembles the shape of a bathtub [26]: Shortly after putting into operation, some products fail at an early stage. For a long subsequent period, the failure rate remains low. Finally, failures agglomerate due to ascending wearout and worse connectivity between components. Having all relevant parameters at hand, questions like this can be answered: Which components form improvable bottlenecks? Is there any evidence for planned obsolescence [11]? From a practical point of view, underlying data represented by a pool of individuals together with temporally staggered events have been made available by logfiles or monitoring reports in tabular form ready for import and analysis. Using action-accumulating P systems, we open a variety of multiset-based tools from membrane computing to a beneficial field of applications in terms of data mining.

When addressing related work, it makes sense to take into consideration different aspects from adjacent fields of research. Basically, our approach adopts some ideas from the *object-oriented paradigm of programming* [20]. Here, individuals called objects carry attributes able to get modified using dedicated methods. Objects can be created and destroyed. Indeed, this paradigm was invented to facilitate abstract modelling of real things for processing on a computer. Nevertheless, objects are typically managed as addressable entities instead of multisets. *Event-based simulation* by means of a queue of events is a well-known technique [2]. In contrast to our approach, events on their own produce further events in the future to keep the system running. Within the *world of P systems*, a loose relation to blotting P systems [15] and population P systems [3, 18] becomes visible due to the consideration of individuals and groups of individuals forming populations. While migration between areas and membranes together with communication among individuals is mainly focussed in population P systems, we emphasise the notion of attributes and events. We are aware of the fact that a setting of attribute values can be interpreted as a location or position referred to a graph-based or nested membrane structure but we believe that attributes are more flexible and more intuitive for our purpose of application.

In Section 2, we familiarise the reader with the formal definition of action-accumulating P systems together with all required prerequisites. Hereafter, two case studies selected from different application scenarios demonstrate its practical use. First, as an introductory example, the popular board game “Mensch ärgere Dich nicht” (Man, don’t get annoyed, a variation of Ludo) is taken under examination in Section 3. Survival, lifetime, and success of pieces running over the board follow specific distributions to be obtained. It is of interest whether a more protective/defensive or a more propulsive strategy offers higher chances to win. The second case study introduced in Section 4 is dedicated to our teaching experiences within the university course “Introduction to Programming” attended by 1,108 students from 2012 to 2016. We identify factors and training strategies suitable for improving the final grade. A final discussion concludes benefits and challenges raising open questions for future work.

2 Action-accumulating P Systems

Formal Prerequisites

Let A and B be arbitrary sets, $\emptyset$ the empty set, $\mathbb{N}$ the set of natural numbers including zero and $\mathbb{R}$ the set of real numbers. A and B are disjoint (share no common elements) iff $A \cap B = \emptyset$. The Cartesian product $A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$ collects all tuples from A and B. For $A \times A$, we write A^2 for short. A Cartesian product might result from more than two sets formalised by n-tuples:

$$\bigtimes_{i=1}^{n} A_i = \{(a_1, \ldots, a_n) \mid a_1 \in A_1 \wedge \ldots \wedge a_n \in A_n\}$$

The term card(A), also written as $|A|$, denotes the number of elements in A (cardinality). $\wp(A)$ symbolises the power set of A containing all $2^{|A|}$ subsets of A as elements. A multiset over A is a mapping $\mathcal{F} : A \longrightarrow \mathbb{N} \cup \{+\infty\}$. Multisets in general can be written as an elementwise enumeration of the form $\{(a_1, \mathcal{F}(a_1)), (a_2, \mathcal{F}(a_2)), \ldots\}$ since $\forall (a, b_1), (a, b_2) \in \mathcal{F} : b_1 = b_2$. A multiset can also be specified by unordered enumeration of multiple elements like for instance $\{a, a, b, a, b\}$ instead of $\{(a, 3), (b, 2)\}$. The support $\mathrm{supp}(\mathcal{F}) \subseteq A$ of $\mathcal{F}$ is defined by $\mathrm{supp}(\mathcal{F}) = \{a \in A \mid \mathcal{F}(a) > 0\}$. A multiset $\mathcal{F}$ over A is said to be empty iff $\forall a \in A : \mathcal{F}(a) = 0$. The cardinality $|\mathcal{F}|$ of $\mathcal{F}$ over A is $|\mathcal{F}| = \sum\limits_{a \in A} \mathcal{F}(a)$.

Let $\mathcal{F}$ and $\mathcal{G}$ be multisets. It holds $\mathcal{F} \subseteq \mathcal{G}$ iff $\mathrm{supp}(\mathcal{F}) \subseteq \mathrm{supp}(\mathcal{G}) \ \wedge \ \forall f \in \mathrm{supp}(\mathcal{F}) \ : \ (\mathcal{F}(f) \leq \mathcal{G}(f))$. Respectively, $\mathcal{F} \subset \mathcal{G}$ iff $\mathrm{supp}(\mathcal{F}) \subseteq \mathrm{supp}(\mathcal{G}) \ \wedge \ \exists f \in \mathrm{supp}(\mathcal{F}) \ : \ (\mathcal{F}(f) < \mathcal{G}(f))$. $\mathcal{F}$ and $\mathcal{G}$ are equal, written as $\mathcal{F} = \mathcal{G}$, iff $\mathcal{F} \subseteq \mathcal{G} \ \wedge \ \mathcal{G} \subseteq \mathcal{F}$.
The multiset union $\mathcal{F} \cup \mathcal{G}$ is defined by
$\mathcal{F} \cup \mathcal{G} = \{(a, h) \mid a \in \mathrm{supp}(\mathcal{F}) \cup \mathrm{supp}(\mathcal{G}) \ \wedge \ h = \max(\mathcal{F}(a), \mathcal{G}(a))\}$,
the multiset intersection $\mathcal{F} \cap \mathcal{G}$ by
$\mathcal{F} \cap \mathcal{G} = \{(a, h) \mid a \in \mathrm{supp}(\mathcal{F}) \cap \mathrm{supp}(\mathcal{G}) \ \wedge \ h = \min(\mathcal{F}(a), \mathcal{G}(a))\}$,
the multiset sum $\mathcal{F} \uplus \mathcal{G}$ by
$\mathcal{F} \uplus \mathcal{G} = \{(a, h) \mid a \in \mathrm{supp}(\mathcal{F}) \cup \mathrm{supp}(\mathcal{G}) \ \wedge \ h = \mathcal{F}(a) + \mathcal{G}(a)\}$,
and the multiset difference $\mathcal{F} \ominus \mathcal{G}$ by
$\mathcal{F} \ominus \mathcal{G} = \{(a, h) \mid a \in \mathrm{supp}(\mathcal{F}) \cup \mathrm{supp}(\mathcal{G}) \ \wedge \ h = \max(\mathcal{F}(a) - \mathcal{G}(a), 0)\}$.

Definition of Systems Components

Let a domain be an arbitrary non-empty set. An action-accumulating P system $\Pi_\Box$ is a construct

$$\Pi_\Box = (C, n, D_1, \ldots, D_n, \mathcal{I}, R, E, m, S_1, \ldots, S_m, s_1, \ldots, s_m)$$

with its components

$C \subseteq \mathbb{N}$ .... domain of points in time (global clock)

$n \in \mathbb{N} \setminus \{0\}$ .... number of distinct attributes

D_i with $i = 1, \ldots, n$ ....domain of attribute i

$\mathcal{I} : \bigtimes\limits_{i=1}^{n} D_i \longrightarrow \mathbb{N} \cup \{+\infty\}$

final multiset of initial individuals in which each individual is represented by the n-tuple of its initial attribute values in conjunction with the number of copies of the individual.

R ....set of actions available for events (see details in the next subsection)

$E \subseteq C \times \wp\left(\left(\bigtimes\limits_{i=1}^{n} D_i\right) \times (\mathbb{N} \cup \{+\infty\})\right) \times R$

final set of events. Each event is described by its point in time $\in C$ followed by the multiset of affected individuals which is a subset of all individuals currently available within the system. Finally, a rule from R expresses the action initiated by the event.

$m \in \mathbb{N} \setminus \{0\}$ .... number of response functions

S_i with $i = 1, \ldots, m$ .... domain of response i

$s_i : \left(\bigtimes_{i=1}^{n} D_i \longrightarrow \mathbb{N} \cup \{+\infty\} \right) \times C \longrightarrow S_i$ with $i = 1, \ldots, m$

response function s_i. Each response function provides an output of the system taking into account the whole cumulative record tracing the evolution of individuals over time starting with the initial setting $\mathcal{I}$ up to the point in time in which all events from E have been processed and the corresponding actions are done. A typical response function might include statistical analysis.

Systems Behaviour and Available Types of Actions

The systems behaviour aims at *tracing* of the individuals together with their attribute values. In order to enable subsequent statistical analysis and interpretation of behavioural patterns, we emphasise the formulation of an algebraic framework able to have at hand the entire systems configuration record comprising the life cycles of all individuals together with the temporal progression of their attribute values. To do so, we define a *transition function*

$$\mathcal{O} : \left(\bigtimes_{i=1}^{n} D_i \right) \times (\mathbb{N} \cup \{+\infty\}) \times \mathbb{N} \longrightarrow \left(\bigtimes_{i=1}^{n} D_i \right) \times (\mathbb{N} \cup \{+\infty\})$$

whose function value collects all individuals (including multiple copies) with their attribute values available in the system at time t. Along with the evolution of the system, $\mathcal{O}(t+1)$ is obtained from $\mathcal{O}(t)$ taking into account all events from E occurring at time t with $t \in C$. Initially, we set

$$\mathcal{O}(0) = \mathcal{I}$$

In case there is *no event* in E at time t, a non-modifying transition is carried out which results in $\mathcal{O}(t+1) = \mathcal{O}(t)$. For all other cases, the transition from $\mathcal{O}(t)$ to $\mathcal{O}(t+1)$ needs to evaluate all events in E defined at time t. We distinguish different types of events which imply specific actions to perform the transition. Since each event from E comes with an action taken from the set R of available actions, we list the corresponding behaviour.

Let $(t, \mathcal{P}, r) \in E$ be an event at time $t \in C$ affecting a multiset of individuals captured by $\mathcal{P} \subseteq \left(\bigtimes_{i=1}^{n} D_i \right) \times (\mathbb{N} \cup \{+\infty\})$ and initiating an action $r \in R$, we specify the action types `modify`, `merge`, `create`, `kill`, and `clone` as follows:

$r = \texttt{modify}\,(\mathrm{f}_1, \ldots, \mathrm{f}_n)$

modifies the attribute values of all individuals from $\mathcal{P}$ using the update functions $\mathrm{f}_i : D_1 \times \ldots \times D_n \longrightarrow D_i$ whereas $i = 1, \ldots, n$. Technically, the transition first removes those individuals from $\mathcal{O}(t)$ whose attribute values have to be renewed followed by addition of corresponding individuals with the updated attribute values:

$$\begin{aligned}\mathcal{O}(t+1) &= \mathcal{O}(t) \ominus \mathcal{V} \uplus \mathcal{W} \text{ with}\\ \mathcal{V} &= \{v \in \mathcal{P} \mid (t, \mathcal{P}, \texttt{modify}(\mathrm{f}_1, \ldots, \mathrm{f}_n))\} \in E\}\\ \mathcal{W} &= \{((\mathrm{f}_1(a_1, ..., a_n), ..., \mathrm{f}_n(a_1, ..., a_n)), \mu) \mid ((a_1, ..., a_n), \mu) \in \mathcal{V}\}\end{aligned}$$

Attention must be paid to the fact that several `modify` actions might take place simultaneously. In order to maintain a deterministic systems behaviour, we require a confluent course in which the sequence of processing the events does not matter. Formally, let $(t, \mathcal{P}_1, \texttt{modify}(\mathrm{f}_1, \ldots, \mathrm{f}_n))$ and $(t, \mathcal{P}_2, \texttt{modify}(\mathrm{g}_1, \ldots, \mathrm{g}_n))$ two concurrent events. It must hold either $\mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset$ (disjoint individuals) or $\mathrm{f}_1(\mathrm{g}_1) = \mathrm{g}_1(\mathrm{f}_1), \ldots, \mathrm{f}_n(\mathrm{g}_n) = \mathrm{g}_n(\mathrm{f}_n)$ for all individuals in $\mathcal{P}_1 \cap \mathcal{P}_2$.

$r = \texttt{merge}(\mathrm{f}_1, \ldots, \mathrm{f}_n)$

removes all individuals in $\mathcal{P}$ and adds one new individual whose initial attribute values are composed from its ancestors in $\mathcal{P}$ using the recombination functions $\mathrm{f}_i : (D_1 \times \ldots \times D_n)^{|\mathcal{P}|} \longrightarrow D_i$ whereas $i = 1, \ldots, n$.

$$\begin{aligned}\mathcal{O}(t+1) &= \mathcal{O}(t) \ominus \mathcal{V} \uplus \mathcal{W} \text{ with}\\ \mathcal{V} &= \{v \in \mathcal{P} \mid (t, \mathcal{P}, \texttt{merge}(\mathrm{f}_1, \ldots, \mathrm{f}_n))\} \in E\}\\ \mathcal{W} &= \{((\mathrm{f}_1(a_{1,1}, ..., a_{|\mathcal{P}|,n}), ..., \mathrm{f}_n(a_{1,1}, ..., a_{|\mathcal{P}|,n})), 1)\}\end{aligned}$$

Analogously to `modify` actions, `merge` actions can influence each other and among others if they run simultaneously. To overcome this ambiguity, we technically process `merge` actions after concurrent `modify` actions. Furthermore, simultaneous `merge` actions are required to be independent from each other or confluent to each other. Formally, let $(t, \mathcal{P}_1, \texttt{merge}(\mathrm{f}_1, \ldots, \mathrm{f}_n))$ and $(t, \mathcal{P}_2, \texttt{merge}(\mathrm{g}_1, \ldots, \mathrm{g}_n))$ be two concurrent events. It must hold either $\mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset$ (disjoint individuals) or $\mathrm{f}_1(\mathrm{g}_1) = \mathrm{g}_1(\mathrm{f}_1), \ldots, \mathrm{f}_n(\mathrm{g}_n) = \mathrm{g}_n(\mathrm{f}_n)$ for all individuals in $\mathcal{P}_1 \cap \mathcal{P}_2$.

$r = \texttt{create}(a_1, \ldots, a_n)$

creates a new individual with initial attribute values $a_1 \in D_1, \ldots, a_n \in D_n$ and adds this individual to the population.

$$\mathcal{O}(t+1) = \mathcal{O}(t) \uplus \{((a_1, \ldots, a_n), 1)\}$$

A `create` action cannot interact with other types of action.

$r =$ `kill`

removes all individuals in $\mathcal{P}$ from the population.

$$\mathcal{O}(t+1) = \mathcal{O}(t) \ominus \mathcal{P}$$

A `kill` action does not influence the resulting composition of the population if other actions occur at the same time.

$r =$ `clone`

duplicates each individual from $\mathcal{P}$ with its attribute values.

$$\mathcal{O}(t+1) = \mathcal{O}(t) \uplus \mathcal{P}$$

It might happen that a `clone` action interferes with one or more `modify` or `merge` actions at the same time. To avoid nondeterminism, we state that `clone` actions get always executed after `modify` and `merge` actions.

In order to finalise the system's description, we still need to mention its response expressed by m freely definable response functions. In this context, we initiate response domains S_1 to S_m capturing all of the system's potential outputs. Based on that, each response function s_i $(i = 1, \ldots, m)$ is allowed to access the entire record $\mathcal{O}(0), \mathcal{O}(1), \mathcal{O}(2), \ldots, \mathcal{O}(\tau)$ of individuals present in the system over time whereas τ marks the temporally last event in E. Formulation of response functions might include statistical tests and/or some dedicated reasoning. Hence, their formal description within application scenarios might become rather extensive.

3 Board Game "Mensch ärgere Dich nicht"

For more than 100 years, people worldwide enjoy the board game "Mensch ärgere Dich nicht" [24], a variation of the English Ludo developed in Germany following the notion of a *cross and circle* game. Its most popular version is made for four players. Here, the board consists of a directed ring forming a cross with 40 places, see Figure 2. Every 10th place serves as a starting position for a player. Directly in front of each starting place is a junction to four consecutive goal places ("safe heaven") of the player according to the starting place. Each player controls four pieces of the same colour. His task is to maneuver all own pieces through the whole cross and place them afterwards inside the safe heaven. The winner is the player who succeeds in placing all of his pieces in the safe heaven before the competitors do. Remaining players continue in a way to successively complete the game on their own. The last one loses the game. The four players act in a cyclic order: black, yellow, green, red. Every player features four pieces initially not in the game but placed outside. The players throw a six-sided dice one after the other. At the beginning and in a situation when all own pieces are outside the game, the player is allowed to throw the dice up to three times. As soon as the dice shows 6 dots, the player sets one of his outside pieces into the game

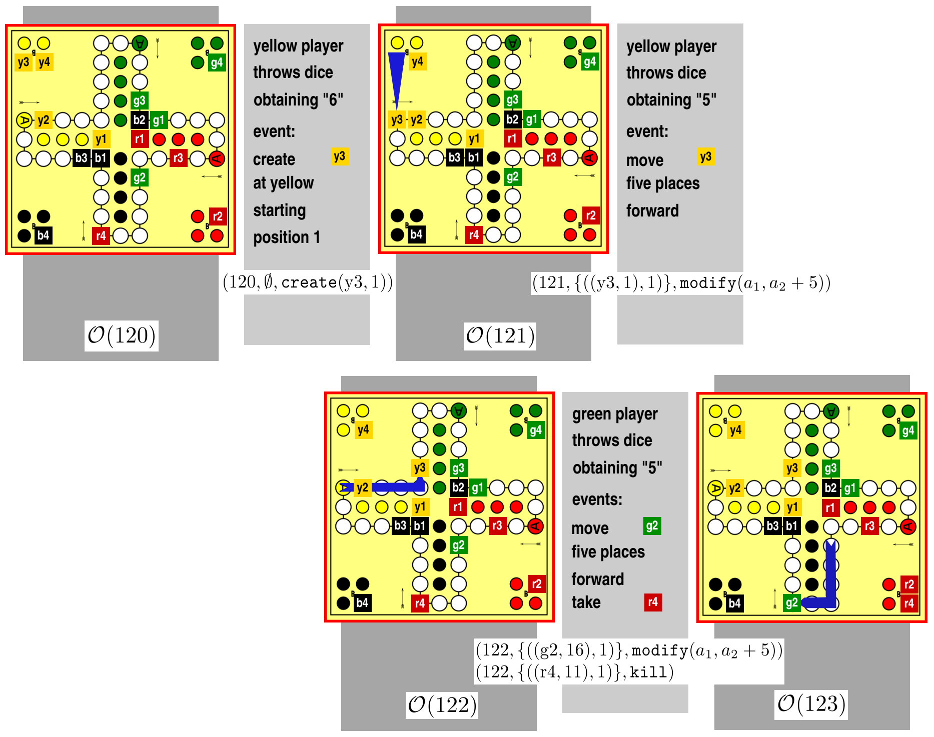

yellow player throws dice obtaining "6" event: create y3 at yellow starting position 1
(120, ∅, create(y3, 1))
𝒪(120)
yellow player throws dice obtaining "5" event: move y3 five places forward
(121, {((y3, 1), 1)}, modify(a1, a2 + 5))
𝒪(121)
green player throws dice obtaining "5" events: move g2 five places forward take r4
(122, {((g2, 16), 1)}, modify(a1, a2 + 5))
(122, {((r4, 11), 1)}, kill)
𝒪(122)
𝒪(123)

Fig. 2. Excerpt of the sample game course after 120 dice throws (upper left configuration of the board). The yellow player sets a new piece ($y3$) into the game and moves it 5 places forward. In the next step, piece $g2$ controlled by the green player reaches the place occupied by piece $r4$ which in turn is taken out of the game. Dice throws along with decisions of the player on which piece to move result in corresponding events depicted informally and in its formal notation.

at the starting position and throws the dice again. After throwing the dice, the player is obliged to move one of his pieces in the game forward along the places by the exact number of dots on top of the dice. If the destination place of a move is occupied by a piece of his own, the move is not allowed. In case it is occupied by an opponent's piece, this one is taken out of the game and has to start the circle anew. Each player is free in choice to select among his pieces to move but two constraints have to be fulfilled: (1) The starting position has to be cleared up with highest priority if there are own pieces outside the game. (2) In case, a piece of another player can be taken out of the game, it is mandatory to do so. After finalising the whole ring, each piece enters its safe heaven. Pieces are not allowed to pass each other inside the safe heaven.

We utilise the toy example "Mensch ärgere Dich nicht" in order to illustrate descriptive and evaluative capabilities of action-accumulating P systems. Based on the common rules of the game together with a comprehensive set of monitoring data from a typical game course, we formulate the system

$$\Pi_{\square} = (C, 2, D_1, D_2, \mathcal{I}, R, E, 4, S_1, s_1, S_2, s_2, S_3, s_3, S_4, s_4)$$

with its components:

$C = \{0, \ldots, 360\}$

Each time the dice is thrown, the global clock performs a tick incrementing the point in time. The sample game ends after 360 time steps.

$D_1 = \{b1, b2, b3, b4, y1, y2, y3, y4, g1, g2, g3, g4, r1, r2, r3, r4\}$

We distinguish a total amount of 16 individual pieces assigned to four players by colour. $b1$ to $b4$ stand for black pieces, $r1$ to $r4$ for red, $y1$ to $y4$ for yellow, and $g1$ to $g4$ for green.

$D_2 = \{0, \ldots, 44\}$

This attribute symbolises the place occupied by a piece. 0 marks the position outside the game. Inside the ring, the places have been consecutively enumerated from the individual piece's starting position 1 up to 40. Values between 41 and 44 identify places within each player's safe heaven.

$\mathcal{I} = \emptyset$

The game starts with an empty population. Pieces act as individuals. Once a piece is set into the game, the corresponding individual gets created. As soon as a piece is taken out of the game, its life cycle finishes. Pieces within the safe heaven remain in the population for the rest of the game course.

$$\begin{aligned} R = {} & \{\texttt{create}(\text{piece}, 1) \mid \text{piece} \in D_1\} \cup \\ & \{\texttt{modify}(a_1, a_2 + d) \mid d \in \{1, \ldots, 6\}\} \ \cup \ \{\texttt{kill}\} \end{aligned}$$

Piece's life cycles might be affected by three types of actions: Individuals enter the game by `create` setting the name of the piece and its starting place 1 as attribute values. Moreover, a move of a piece is reflected by `modify` in which the value of its second attribute a_2 gets increased by the number d shown on top of the dice.

$$\begin{aligned} E = \{ & (0, \emptyset, \texttt{create}(b1, 1)), \\ & (1, \{((b1, 1), 1)\}, \texttt{modify}(a_1, a_2 + 3)), \\ & (6, \emptyset, \texttt{create}(g1, 1)), \\ & (7, \{((g1, 1), 1)\}, \texttt{modify}(a_1, a_2 + 2)), \\ & (11, \{((b1, 4), 1)\}, \texttt{modify}(a_1, a_2 + 1)), \\ & (15, \{((g1, 3), 1)\}, \texttt{modify}(a_1, a_2 + 2)), \\ & \quad \vdots \end{aligned}$$

$$\begin{aligned}
&(120, \emptyset, \texttt{create}(y3, 1)),\\
&(121, \{((y3, 1), 1)\}, \texttt{modify}(a_1, a_2 + 5)),\\
&(122, \{((g2, 16), 1)\}, \texttt{modify}(a_1, a_2 + 5)),\\
&(122, \{((r4, 11), 1)\}, \texttt{kill}),\\
&\quad\vdots\\
&(357, \{((b4, 36), 1)\}, \texttt{modify}(a_1, a_2 + 2)),\\
&(358, \{((g3, 4), 1)\}, \texttt{modify}(a_1, a_2 + 5)),\\
&(359, \{((b4, 38), 1)\}, \texttt{modify}(a_1, a_2 + 3))\ \}
\end{aligned}$$

Our sample game course comprises $|E| = 379$ events in total. In 32 cases, a piece is taken out of the game by `kill`.

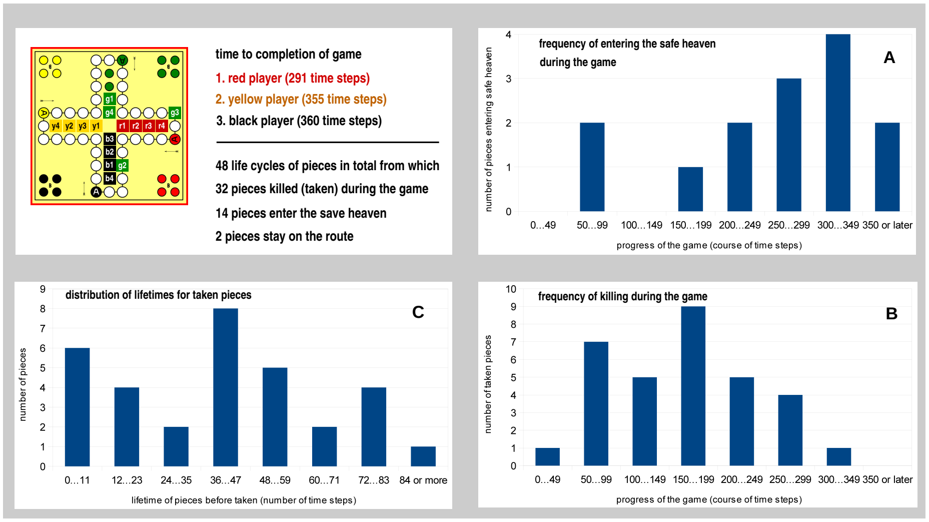
Before finalisation of system's specification by four response domains S_1 to S_4 and dedicated response functions s_1 to s_4 for evaluation of the game, we illustrate the system's behaviour by means of the transition function over time. Inspired by the idea to attract more interest in the overall game course, the players follow different strategies. *Red* acts in a highly propulsive manner forcing one piece to run the ring as fast as possible while its further pieces reside outside the game or near the starting place. A more moderate but still propulsive strategy is exhibited by the *yellow* player who avoids setting its top piece onto places with a high risk of danger. The strategy of the *black* player is dominated by a defensive but present style of action. Keeping many pieces inside the game and close to each other, this strategy aims at mutual protection of pieces. In contrast to all others, the *green* "infant" player moves its pieces in a more or less naive way without any visible strategy.

By evolution of the transition function $\mathcal{O}$ processing the events from E, we obtain the configuration of the game successively for all points in time:

$$\begin{aligned}
\mathcal{O}(0) &= \emptyset\\
\mathcal{O}(6) &= \mathcal{O}(5) = \mathcal{O}(4) = \mathcal{O}(3) = \mathcal{O}(2) = \mathcal{O}(1) = \{((b1, 1), 1)\}\\
\mathcal{O}(7) &= \{((b1, 4), 1), ((g1, 1), 1)\}\\
\mathcal{O}(8) &= \{((b1, 4), 1), ((g1, 3), 1)\}\\
&\ \ \vdots\\
\mathcal{O}(120) &= \{((b1, 5), 1), ((b2, 25), 1), ((b3, 6), 1), ((y1, 44), 1), ((y2, 2), 1), ((g1, 6), 1),\\
&\quad ((g2, 16), 1), ((g3, 4), 1), ((r1, 44), 1), ((r3, 3), 1), ((r4, 11), 1)\}\\
&\ \ \vdots\\
\mathcal{O}(360) &= \{((b1, 42), 1), ((b2, 43), 1), ((b3, 44), 1), ((b4, 41), 1), \ldots, ((r4, 41), 1)\}
\end{aligned}$$

Evaluation 1: Identification of the ranking among all players

Having the transition function at hand, we can figure out the ranking of all players in the game. The ranking is determined by the point in time in which the last piece of a player enters its safe heaven. To this end, we formulate all necessary constraints to express the corresponding response function:


time to completion of game

1. red player (291 time steps)

2. yellow player (355 time steps)

3. black player (360 time steps)

48 life cycles of pieces in total from which

32 pieces killed (taken) during the game

14 pieces enter the save heaven

2 pieces stay on the route

Fig. 3. Board game "Mensch ärgere Dich nicht", condensed representation of evaluation results. Upper left part: final configuration of the game captured by $\mathcal{O}(360)$ and ranking of the players. Part **A**: frequency of entering safe heavens during the game corresponding to evaluation 2. Part **B**: frequency of taking out figures during the game (evaluation 3). Part **C**: distribution of lifetimes for taken pieces (evaluation 4).

$$
\begin{aligned}
&S_1 = C \\
&s_1 \;:\; \{b, y, g, r\} \longrightarrow S_1 \\
&s_1 = \{(b, t_b), (y, t_y), (g, t_g), (r, t_r) \mid \\
&\qquad \exists t_b \in C.\forall p \in \{b1, b2, b3, b4\}.[((p, z), 1) \in \mathcal{O}(t_b) \wedge (z > 40) \wedge ((p, z), 1) \notin \mathcal{O}(t_b - 1)] \vee \\
&\qquad \exists t_y \in C.\forall p \in \{y1, y2, y3, y4\}.[((p, z), 1) \in \mathcal{O}(t_y) \wedge (z > 40) \wedge ((p, z), 1) \notin \mathcal{O}(t_y - 1)] \vee \\
&\qquad \exists t_g \in C.\forall p \in \{g1, g2, g3, g4\}.[((p, z), 1) \in \mathcal{O}(t_g) \wedge (z > 40) \wedge ((p, z), 1) \notin \mathcal{O}(t_g - 1)] \vee \\
&\qquad \exists t_r \in C.\forall p \in \{r1, r2, r3, r4\}.[((p, z), 1) \in \mathcal{O}(t_r) \wedge (z > 40) \wedge ((p, z), 1) \notin \mathcal{O}(t_r - 1)]\}
\end{aligned}
$$

We obtain $s_1 = \{(b, 360), (y, 355), (r, 291)\}$. Thus, the red player was the winner followed by the yellow and then by the black player. The green player lost the game since he failed in placing all of his pieces in the save heaven. The ranking gives evidence to hypothesise that a highly propulsive strategy could be a promising way. Indeed, after performing an amount of 10 games, the most propulsive player succeeded in winning 8 times. A defensive strategy commonly prevents the player from losing the game but also from the winner state.

Evaluation 2: Frequency of entering safe heavens during the game

In this evaluation, we would like to answer the question about the distribution of arrivals of the pieces in the safe heavens during the game course. Obviously, from the beginning of the game it needs a certain amount of steps before the first piece reaches its final destination. The response function denoted as a multiset (allowed to be written as a set containing multiple copies of elements) reads as follows:

$$\begin{aligned}
&S_2 = \mathbb{N}\\
&s_2 \;:\; \{p_0, \ldots, p_{360}\} \longrightarrow S_2\\
&s_2 = \{p_{enter} \mid \exists enter \in C \;.\; \exists y, z \in D_2 \;.\; \exists x \in D_1 \;.[((x,y),1) \in \mathcal{O}(enter) \wedge (y > 40) \wedge\\
&\qquad ((x,z),1) \in \mathcal{O}(enter-1) \wedge (z \leq 40) \wedge \forall t \in C \text{ with } (t > enter) \;.\; [((x,\alpha),1) \in \mathcal{O}(t)]]
\end{aligned}$$

Altogether, 14 out of 16 pieces in total finally reside inside one of the safe heavens. Response function $s_2 = \{p_{96}, p_{99}, p_{199}, p_{220}, p_{244}, p_{253}, p_{259}, p_{291}, p_{309}, p_{316}, p_{347}, p_{349}, p_{355}, p_{360}\}$ identifies all points in time in which a piece enters a safe heaven. It turns out that the majority of pieces arrives during the second half of the game course with the highest frequency in the last quarter. Figure 3A shows a more condensed view of the frequency distribution.

Evaluation 3: Frequency of killing during the game

In this study, we would like to know in which phase of the game most pieces get killed. To this end, we elaborate a distribution of the according frequencies over the game course resulting in the response function denoted as a multiset:

$$\begin{aligned}
&S_3 = \mathbb{N}\\
&s_3 \;:\; \{p_0, \ldots, p_{360}\} \longrightarrow S_3\\
&s_3 = \{p_{end} \mid \exists begin \in C \;.\; \exists end \in C \;.\; \exists y \in D_2 \;.\; \exists z \in D_2 \;.\; \exists x \in D_1 \;.\\
&\qquad [((x,y),1) \in \mathcal{O}(begin) \wedge ((x,y),1) \notin \mathcal{O}(begin-1) \wedge ((x,z),1) \in \mathcal{O}(end) \wedge\\
&\qquad ((x,z),1) \notin \mathcal{O}(end+1) \wedge (y > 0) \wedge (y \leq 40) \wedge (z > 0) \wedge (z \leq 40) \wedge (z \geq y) \wedge\\
&\qquad (\forall w \in \{begin, \ldots, end\} \;.\; [((x,\alpha),1) \in \mathcal{O}(w) \wedge (\alpha > 0) \wedge (\alpha \leq 40)])]\}
\end{aligned}$$

This evaluation categorises the life cycles of pieces whose lifetime terminates before the end of the game. For this purpose, the *trace* of prematurely killed individuals through the transition function has to be identified. Obviously, the life of a piece begins when set into the game. Prior to this point in time indicated by *begin*, the piece is not present. The death of this piece (taking out of the game) also occurs at a certain point in time called *end*. In between, the piece must persist without any interruption. Additionally, we require that the life cycle of a piece ends within the ring (places 1 to 40). After processing the evaluation, we obtain: $s_3 = \{p_{36}, p_{56}, p_{58}, p_{59}, p_{73}, p_{81}, p_{93}, p_{99}, p_{121}, p_{127}, p_{128}, p_{135}, p_{137}, p_{157}, p_{158}, p_{165}, p_{166}, p_{171}, p_{180}, p_{181}, p_{189}, p_{192}, p_{210}, p_{219}, p_{223}, p_{224}, p_{248}, p_{264}, p_{277}, p_{280}, p_{295}, p_{304}\}$. In total, 32 pieces get killed during the game. It stands out that pieces lose their life most frequently around half of the game. Within the first fifth and within the last fifth of the game, merely few pieces are taken. Figure 3B subsumes the underlying distribution in condensed form.

Evaluation 4: Lifetime distribution of killed pieces

In addition to evaluation 3, we focus on the distribution of age reached by those pieces killed during the game. This lifetime distribution resembles the shape of a bathtub in many real-world scenarios since a number of individuals dies shortly after birth while most individuals survive for a long time before they die as well. In our study, the lifetime distribution results from following response function denoted as a multiset by enumeration of its (multiple) elements:

$$\begin{aligned}
&S_4 = \mathbb{N}\\
&s_4 \ : \ \{p_0, \ldots, p_{360}\} \longrightarrow S_4\\
&s_4 = \{p_{end-begin} \mid \exists begin \in C \ . \ \exists end \in C \ . \ \exists y \in D_2 \ . \ \exists z \in D_2 \ . \ \exists x \in D_1 \ .\\
&\qquad [((x,y),1) \in \mathcal{O}(begin) \wedge ((x,y),1) \notin \mathcal{O}(begin-1) \wedge ((x,z),1) \in \mathcal{O}(end) \wedge\\
&\qquad ((x,z),1) \notin \mathcal{O}(end+1) \wedge (y>0) \wedge (y \leq 40) \wedge (z>0) \wedge (z \leq 40) \wedge (z \geq y) \wedge\\
&\qquad (\forall w \in \{begin, \ldots, end\} \ . \ [((x,\alpha),1) \in \mathcal{O}(w) \wedge (\alpha > 0) \wedge (\alpha \leq 40)])]\}
\end{aligned}$$

Analogously to evaluation 3, for each piece x the time point of "birth" has been identified by *begin* and the time point of "killing" by *end*, respectively. Moreover, we select those individuals taken inside the ring which means that their attribute value for the place must be between 1 and 40. Furthermore, we demand that every piece permanently exists from its birth to its death without any interruption. The outcome discloses the desired lifetime distribution taking into account all 32 prematurely killed pieces: $s_4 = \{p_4, p_5, p_6, p_8, p_9, p_{11}, p_{14}, p_{15}, p_{16}, p_{17}, p_{33}, p_{34}, p_{37}, p_{37}, p_{38}, p_{38}, p_{39}, p_{40}, p_{41}, p_{44}, p_{52}, p_{54}, p_{55}, p_{56}, p_{57}, p_{61}, p_{69}, p_{72}, p_{74}, p_{79}, p_{80}, p_{94}\}$. In contrast to the bathtub-shape, we observe three instead of two peaks due to the fact that pieces preferably get killed near the starting places of opponent players. For a more condensed illustrative representation of the lifetime distribution, see Figure 3C.

4 University Course "Introduction to Programming"

A basic knowledge in computer programming and software development belongs to essential skills in many disciplines of engineering, natural, and life sciences. Thus, university courses addressing an introduction to programming using a popular programming language like Java have been regularly offered at more than 100 public universities in Germany. In this case study, we analyse the one-semester course held by us eight times within the period from autumn 2012 to summer 2016 having sole responsibility. In total, 1108 students attended this course from which 808 finally got graded. Each student participating in this course represents an individual whose life cycle indicates 10 consecutive *phases* of the course passed over 18 weeks. After the initial *enrolment* at the beginning of the semester, two lectures per week take place flanked by exercises and practical trainings in small groups. Approximately every two weeks, a written report

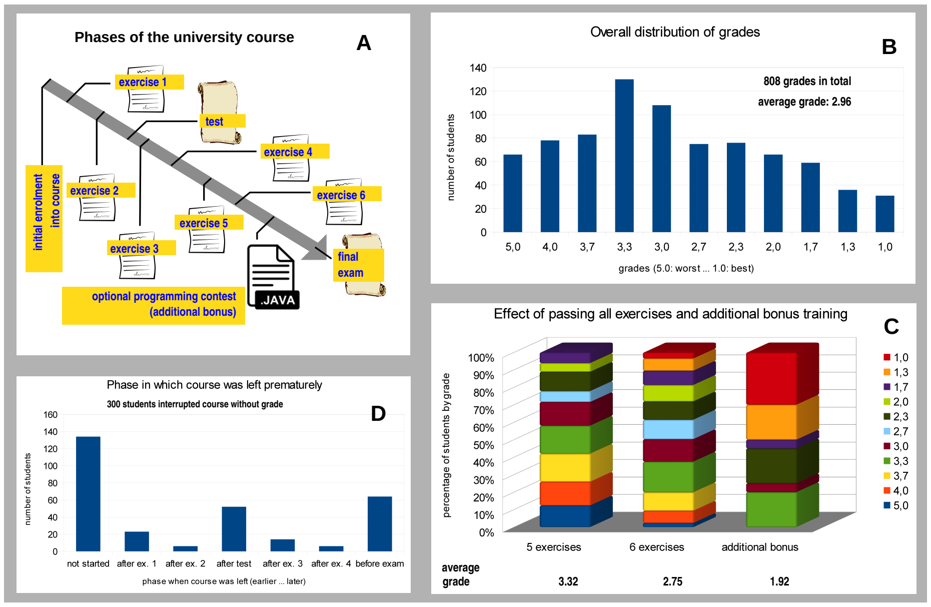


Fig. 4. Summary of evaluations for our one-semester university course "Introduction to Programming" held eight times from autumn 2012 to summer 2016. **A**: phases of the course, **B**: overall distribution of grades, **C**: effect of passing all exercises and additional bonus training, **D**: frequency of phases in which course was left prematurely

with solutions and self-made source code to diverse tasks needs to be submitted whereas an annotated feedback about success (passed / failed) is given afterwards. Altogether, *six* separate submissions according to the topics of exercises are planned from which at least five have to be passed to finalise the course. In addition, around midterm a rated written *test* is carried out in which up to 30 scores can be reached. At the end of the lecture period, the students can take part in an optional programming contest for gaining an additional *bonus* of up to five scores if appropriate. At last, the final written *exam* is mandatory to graduate from the course. Here, up to 70 scores are available. Based on the individual number of scores in total $(0, \ldots, 105)$ in conjunction with the number of passed exercise submissions $(0, \ldots, 6)$, the final *grade* has been assigned. To this end, according to the common German classification system, the grades ordered from best to worst cover 1.0, 1.3 (excellent level), 1.7, 2.0, 2.3 (good), 2.7, 3.0, 3.3 (satisfactory), 3.7, 4.0 (fair), and 5.0 (fail). Beyond various informal and subjective quality indicators, a cumulative statistical evaluation taking into account pseudonymised quantifiable data from the participating students can support a comparison between different course offers among competing universities or departments. Having at hand the underlying recorded data organised into the structure of events associated with the predefined phases of the course, we create the corresponding action-accumulating P system as follows:

$$\Pi_\square = (C, 11, D_1, \ldots, D_{11}, \mathcal{I}, R, E, 3, S_1, s_1, S_2, s_2, S_3, s_3)$$

with its components:

$C = \{0, \ldots, 9\}$

Points in time reflecting the consecutive phases of the course such as 0: enrolment; $1, 2$: submission to exercise one and two; 3: midterm test, $4, 5, 6, 7$: submission to exercise three, four, five, and six; 8: optional programming contest for additional bonus; 9: final examination and grade.

$D_1 = (\{\mathrm{A}, \ldots, \mathrm{Z}\} \cup \{0, \ldots, 9\})^*$

Finite sequence of characters acting as pseudonym that identifies an individual. It uniquely results from the student's ID along with an indicator of the semester in which the course was attended (like summer term 2016) and a random number. A pseudonym is allowed to be empty.

$D_2 = D_3 = D_4 = D_5 = D_6 = D_7 = \{0, 1\}$

For each exercise with report submission in ascending order, the corresponding attribute indicates either passed (1) or failed/skipped (0).

$D_8 = \{0, \ldots, 30\}$

Number of scores reached in the midterm test.

$D_9 = \{0, \ldots, 5\}$

Number of scores reached in the optional programming contest (additional bonus).

$D_{10} = \{0, \ldots, 70\}$

Number of scores reached in the final examination.

$D_{11} = \{1.0, 1.3, 1.7, 2.0, 2.3, 2.7, 3.0, 3.3, 3.7, 4.0, 5.0\} \cup \{\infty\}$

Range of grades. The auxiliary symbol ∞ stands for no grade.

$$\mathcal{I} = \{((326\mathrm{C}638, 0, 0, 0, 0, 0, 0, 0, 0, 0, \infty), 1), \ldots, \\ ((2\mathrm{F}56771, 0, 0, 0, 0, 0, 0, 0, 0, 0, \infty), 1)\}$$

Multiset of enrolled student's individuals, each at the begin of the course.

$$\begin{aligned} R = \{&\texttt{create}(d_1, \ldots, d_{11}) \mid d_1 \in D_1 \wedge \ldots \wedge d_{11} \in D_{11}\} \;\cup\; \{\texttt{kill}\} \;\cup\; \{\texttt{clone}\} \;\cup \\ \{&\texttt{modify}(d_1, d_2 + e_1, \ldots, d_7 + e_6, z, d_9 + b, p, g) \mid e_1 \in D_2 \wedge \ldots \wedge e_6 \in D_7 \wedge \\ &\qquad z \in D_8 \wedge b \in D_9 \wedge s \in D_{10} \wedge g \in D_{11})\} \;\cup \\ \{&\texttt{merge}(\bigotimes_{\substack{d_1 \text{ with} \\ (d_1, \ldots, d_{11}) \in \mathcal{P}}} d_1, \sum_{\substack{d_2 \text{ with} \\ (d_1, \ldots, d_{11}) \in \mathcal{P}}} d_2, \ldots, \sum_{\substack{d_7 \text{ with} \\ (d_1, \ldots, d_{11}) \in \mathcal{P}}} d_7, 0, 0, 0, \infty)\} \end{aligned}$$

Events of type `create` enable addition of individuals joining the course late. By means of `modify`, the numerical attributes might get updated after each phase of the course. Individuals who leave the course prematurely imply `kill` events. The `merge` event allows for unification of individuals representing the same student attending the course more than once in a line of semesters. Here, $\otimes$ symbolises successive concatenation of underlying pseudonymous character strings. Events of type `clone` have been induced when a student continues the course after interruption one or more semesters later. In this case, the partial fulfilment of selected course phases can be approved by maintaining the corresponding attribute values.

$$\begin{aligned}E = \{&(1, \{((342\text{D}5\text{B}8, 0, 0, 0, 0, 0, 0, 0, 0, 0, \infty), 1)\}, \texttt{modify}(d_1, d_2 + 1, d_3, \ldots, d_{11})),\\ &\vdots\\ &(9, \{((3356\text{B}8, d_2, ..., d_{10}, \infty), 1) \mid d_i \in D_i \wedge i = 2, ..., 10\}, \texttt{modify}(d_1, ..., d_{10}, 1.7))\}\end{aligned}$$

The entire set of events consists of $7,219$ entries resulting from eight editions of the course.

Based on the aforementioned $\Pi_\square$ components, the transition function $\mathcal{O}$ is determined revealing all system's configurations from $\mathcal{O}(0)$ to $\mathcal{O}(10)$. Now, we turn to three evaluation studies suitable to shed light on the course attractiveness for students and a potential for improvement of grades.

Evaluation 1: Overall distribution of grades

For gaining the overall distribution of grades, we define at first an auxiliary index function $\mathrm{g} : \{1, \ldots, 11\} \longrightarrow D_{11} \setminus \{\infty\}$ with $\mathrm{g}(1) = 1.0$, $\mathrm{g}(2) = 1.3$, $\mathrm{g}(3) = 1.7$, $\mathrm{g}(4) = 2.0$, $\mathrm{g}(5) = 2.3$, $\mathrm{g}(6) = 2.7$, $\mathrm{g}(7) = 3.0$, $\mathrm{g}(8) = 3.3$, $\mathrm{g}(9) = 3.7$, $\mathrm{g}(10) = 4.0$, and $\mathrm{g}(11) = 5.0$. Using this function, we can denote the response function s_1 representing a multiset whose elements may appear in multiple copies:

$$\begin{aligned}&S_1 = \mathbb{N}\\ &s_1 \;:\; D_{11} \setminus \{\infty\} \longrightarrow S_1\\ &s_1 = \{\mathrm{g}(i) \mid \exists x \in D_1 \;.\; \exists d_2 \in D_2 ... \exists d_{10} \in D_{10} \;.\; \exists \mathit{grade} \in D_{11} \setminus \{\infty\} \;.\; \exists i \in \{1, ..., 11\} \;.\\ &\qquad [(((x, d_2, ..., d_{10}, \mathit{grade}), 1) \in \mathcal{O}(9)) \wedge (\mathit{grade} = \mathrm{g}(i)) \wedge \left(\sum_{k=2}^{7} d_k \geq 5\right)]\}\end{aligned}$$

The evaluation result discloses the distribution of grades whose shape resembles a bell according to a Gaussian curve of distribution which is typical for the spread of talents, diligence, and motivation. The outcome of s_1 in detail reads: $s_1 = \{(5.0, 66), (4.0, 78), (3.7, 83), (3.3, 130), (3.0, 108), (2.7, 75), (2.3, 76), (2.0, 66), (1.7, 59), (1.3, 36), (1.0, 31)\}$. As a consequence, 742 out of 808 attendees – which is approximately 91.8% – taking part in the final examination successfully passed the course but merely 67 students (ca. 8.3%) graduated with an excellent mark. For a graphical illustration, see Figure 4B. The overall average grade is:

$$avg = \frac{\sum_{i=1}^{11} \mathrm{g}(i) \cdot s_1(\mathrm{g}(i))}{|s_1|} \approx 2.96$$

Evaluation 2: Impact of extensive training

It seems to be obvious that achievement of the best possible grade is closely related with the amount of practical training regardless of intrinsic factors like talent. Now, we disclose the impact of passing *all* six exercises and the optional programming contest for additional bonus. To this end, we divide the pool of individuals into *three* disjoint groups such that the first one (secondary index 1) contains all students passed the minimum number of five exercises without any significant additional bonus. The second group (2) comprises all students managed to finalise all six exercises, and the third group (3) in addition attained three or more bonus credits in the optional programming contest. We make use of the same auxiliary index function g introduced in the previous evaluation.

$$\begin{aligned}
&S_{2,1} = \mathbb{N}\\
&s_{2,1} \ : \ D_{11} \setminus \{\infty\} \longrightarrow S_{2,1}\\
&s_{2,1} = \{\mathrm{g}(i) \mid \exists x \in D_1 \ . \ \exists d_2 \in D_2 ... \exists d_{10} \in D_{10} \ . \ \exists grade \in D_{11} \setminus \{\infty\} \ . \ \exists i \in \{1, ..., 11\} \ .\\
&\qquad [(((x, d_2, ..., d_{10}, grade), 1) \in \mathcal{O}(9)) \wedge (grade = \mathrm{g}(i)) \wedge \left(\sum_{k=2}^{7} d_k = 5\right) \wedge (d_9 \leq 2)]\}
\end{aligned}$$

$$\begin{aligned}
&S_{2,2} = \mathbb{N}\\
&s_{2,2} \ : \ D_{11} \setminus \{\infty\} \longrightarrow S_{2,2}\\
&s_{2,2} = \{\mathrm{g}(i) \mid \exists x \in D_1 \ . \ \exists d_2 \in D_2 ... \exists d_{10} \in D_{10} \ . \ \exists grade \in D_{11} \setminus \{\infty\} \ . \ \exists i \in \{1, ..., 11\} \ .\\
&\qquad [(((x, d_2, ..., d_{10}, grade), 1) \in \mathcal{O}(9)) \wedge (grade = \mathrm{g}(i)) \wedge \left(\sum_{k=2}^{7} d_k = 6\right) \wedge (d_9 \leq 2)]\}
\end{aligned}$$

$$\begin{aligned}
&S_{2,3} = \mathbb{N}\\
&s_{2,3} \ : \ D_{11} \setminus \{\infty\} \longrightarrow S_{2,3}\\
&s_{2,3} = \{\mathrm{g}(i) \mid \exists x \in D_1 \ . \ \exists d_2 \in D_2 ... \exists d_{10} \in D_{10} \ . \ \exists grade \in D_{11} \setminus \{\infty\} \ . \ \exists i \in \{1, ..., 11\} \ .\\
&\qquad [(((x, d_2, ..., d_{10}, grade), 1) \in \mathcal{O}(9)) \wedge (grade = \mathrm{g}(i)) \wedge \left(\sum_{k=2}^{7} d_k \geq 5\right) \wedge (d_9 > 2)]\}
\end{aligned}$$

It clearly turns out that the frequency of better grades increases with more practical training. Figure 4C subsumes the outcome. Corresponding average grades for the groups $k = 1, 2, 3$ by $avg_k = \frac{\sum_{i=1}^{11} \mathrm{g}(i) \cdot s_{2,k}(\mathrm{g}(i))}{|s_{2,k}|}$ constitute $avg_1 \approx 3.32$, $avg_2 \approx 2.75$, $avg_3 \approx 1.92$ which results roughly spoken in a tremendous *improvement of two stages* from one group to the next one. Training matters.

Evaluation 3: Phase in which course was left prematurely

From 1,108 students in total, 808 got graded which implies a difference of exactly 300 individuals who left the course prematurely. In this study, we want to clarify in which phase this predominantly happens in order to propose adequate counteractions and stimuli. Formally, for each relevant phase of the course we accumulate the number of students dropping out directly afterwards by s_3.

$$\begin{aligned}
&S_3 = \mathbb{N} \\
&s_3 \ : \ \{p_1, \ldots, p_8\} \longrightarrow S_3 \\
&s_3 = \{p_t \mid \exists t \in C \ . \ \forall i \in \{1, ..., 10\} \ . \ [\exists h_i \in D_{i+1}] \ . \ [(((x, h_1, ..., h_{10}), 1) \in \mathcal{O}(t)) \wedge \\
&\qquad (h_{10} = \infty) \wedge (h_t > 0) \wedge (\forall \tau \in \{t+1, ..., 9\} \, . \, [(((x, l_2, ..., l_{11}), 1) \in \mathcal{O}(\tau)) \wedge (l_\tau = 0)])]\}
\end{aligned}$$

Figure 4D shows the resulting distribution. By detailed inspection of $s_3 = \{(p_1, 134), (p_2, 23), (p_3, 6), (p_4, 52), (p_5, 14), (p_6, 6), (p_7, 1), (p_8, 64)\}$ it becomes visible that 134 students enrolled but were absent from the beginning. 52 students left after the midterm test, and 64 students passed enough exercises but did not take part in the final examination. The number of students leaving the course in between is negligible.

5 Conclusions

Our concept of action-accumulating P systems is mainly motivated by interdisciplinary applications in data mining, data science, and information retrieval which benefits from the growing availability of logfiles and time-stamped data records. Gaining insights into hidden laws of life cycles and generalised behavioural patterns within complex systems exclusively from observed events promises a fascinating field opened for membrane computing. Particularly, the combination of multiset-based modelling with evaluation techniques incorporating predicate logic turns out to be an exploitably powerful tool to cope with descriptive and inferential statistics for bridging empirical knowledge with scientific expressiveness. We believe that our proposed framework of action-accumulating P systems provides a flexible toolbox able to be efficiently adopted in numerous scenarios. Future work will be directed at extension of our approach by *dynamical attributes* which might arise or disappear over time triggered by dedicated events.

References

1. A. Alhazov, S. Cojocaru, A. Colesnicov, L. Malahova, M. Petic. A P System for Annotation of Romanian Affixes. *Lecture Notes in Computer Science* **8340**:80-87, 2014
2. J. Banks (ed.) *Handbook of Simulation.* John Wiley & Sons, 1998
3. F. Bernardini, M. Gheorghe. Population P Systems. *Journal of Universal Computer Science* **10(5)**:509-539, 2004

4. A. Broggi, M. Buzzoni, S. Debattisti, P. Grisleri, M. C. Laghi, P. Medici, P. Versari. Extensive Tests of Autonomous Driving Technologies. *IEEE Transactions on Intelligent Transportation Systems* **14(3)**:1403-1415, 2013
5. M.E. Bakir, M. Gheorghe, S. Konur, M. Stannett. Comparative Analysis of Statistical Model Checking Tools. *Lecture Notes in Computer Science* **10105**:119-135, 2017
6. H. Chen, R.H.L. Chiang, V.C. Storey. Business Intelligence and Analytics: From Big Data to Big Impact. *MIS Quarterly* **36(4)**:1165-1188, 2012
7. A. Ciobanu, F. Ipate. Implementation of P Systems by Using Big Data Technologies. *Lecture Notes in Computer Science* **8340**:117-137, 2014
8. F. Cunha, J.J. Heckman, L.J. Lochner, D.V. Masterov. *Interpreting the evidence on life cycle skill formation.* In Handbook of the Economics of Education, chapter 12, pp. 697-812, Elsevier, 2006
9. S. Erevelles, N. Fukawa, L. Swayne. Big Data consumer analytics and the transformation of marketing. *Elsevier Journal of Business Research* **69(2)**:897-904, 2016
10. A. Estrin, M. Kaminski. The Expressive Power of Temporal Logic of Actions. *Journal of Logic and Computation* **12(5)**:839-859, 2002
11. P.A. Grout, I.U. Park. Competitive planned obsolescence. *RAND Journal of Economics* **36(3)**:596-612, 2005
12. J.B. Guinee. Handbook on life cycle assessment operational guide to the ISO standards. *The International Journal of Life Cycle Assessment* **7(3)**:158-166, 2002
13. J. Han, M. Kamber, J. Pei. *Data Mining Concepts and Techniques.* Morgan Kaufmann and Elsevier, 2012
14. M. Hilbert, P. Lopez. The Worlds Technological Capacity to Store, Communicate, and Compute Information. *Science* **332**:60-65, 2011
15. T. Hinze, K. Grützmann, B. Höckner, P. Sauer, S. Hayat. Categorised Counting Mediated by Blotting Membrane Systems for Particle-Based Data Mining and Numerical Algorithms. *Lecture Notes in Computer Science* **8961**:241-257, 2014
16. T. Hinze, L. Weber, U. Hatnik. Walking Membranes: Grid-Exploring P Systems with Artificial Evolution for Multi-purpose Topological Optimisation of Cascaded Processes. *Lecture Notes in Computer Science* **10105**:251-271, 2017
17. Y. Jiang, H. Peng, X. Huang, J. Zhang, P. Shi. A Novel Clustering Algorithm Based on P Systems. *International Journal of Innovative Computing, Information and Control* **10(2)**:753-765, 2014
18. P. Kefalas, I. Stamatopoulou, G. Eleftherakis, M. Gheorghe. Transforming State-Based Models to P Systems Models in Practice. *Lecture Notes in Computer Science* **5391**:260-273, 2009
19. L. Lamport. The temporal logics of actions. *ACM Transactions on Programming Languages and Systems* **16(3)**:872-923, 1994
20. B. Meyer. *Object-oriented software construction.* Prentice Hall, 1997
21. D.E. O'Leary. *Enterprise Resource Planning Systems: Systems, Life Cycle, Electronic Commerce, and Risk* Cambridge University Press, 2000
22. L. Neugarten. Time, age, and the life cycle. *The American Journal of Psychiatry* **136(7)**:887-894, 1979
23. G.L. Stüber. *Principles of Mobile Communication.* Springer Verlag, 2011
24. F. Wallhoff, A. Bannat, J. Gast, T. Rehrl, M. Dausinger, G. Rigoll. Statistics-Based Cognitive Human-Robot Interfaces for Board Games – Let's Play! *Lecture Notes in Computer Science* **5618**:708-715, 2009
25. I.H. Witten, E. Frank, M.A. Hall, C.J. Pal. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann and Elsevier, 2017
26. G. Yang. *Life Cycle Reliability Engineering.* John Wiley & Sons, 2007

On Evolution-Communication P systems with Energy Having Bounded and Unbounded Communication

Richelle Ann B. Juayong, Nestine Hope S. Hernandez,
Francis George C. Cabarle, Kelvin C. Buño, Henry N. Adorna

Algorithms & Complexity Lab
Department of Computer Science
University of the Philippines Diliman
Diliman 1101 Quezon City, Philippines
E-mail: {rbjuayong,fccabarle,hnadorna}@up.edu.ph,
{nshernandez,kcbuno}@dcs.upd.edu.ph

Abstract. We explore the computing power of Evolution-Communication P systems with energy (ECPe systems) considering dynamical communication measures, $ComN$, $ComR$ and $ComW$. These measures consider the number of communication steps, communication rules and total energy used per communication step, respectively. In this paper, we address a previous conjecture that states that only semilinear sets can be generated with bounded $ComX$, $X \in \{N, R, W\}$. Our result on bounded $ComW$ seems to support such conjecture while the conjecture is not true for bounded $ComN$ and $ComR$. We also show that the class of recursively enumerable sets can be computed using ECPe systems with two membranes. This improves a previous result that makes use of four membranes to show computational completeness.

Keywords: Membrane computing, Evolution-Communication P systems with Energy, Dynamical communication measures

1 Introduction

Evolution-Communication P systems with energy (ECPe systems) is proposed in [1] as a model for studying communication complexity in a membrane system. ECPe system is similar to a variant called Evolution-Communication P systems (ECP systems) introduced in an earlier work [3]. These models are interesting variants for analyzing communication since distinct forms of rules are utilized for evolution and communication. However, in ECPe systems, a special object e, for energy, is presented as a form of communication cost. These e's can be earned or produced during evolution and consumed as requirement for every communication.

An approach adapted from classical communication complexity [11] was proposed to introduce a different way of analyzing communication in [1]. Specifically, *dynamical communication measures* were presented. For every successful computation, these measures consider the number of communication steps (or $ComN$), total number of communication rules (or $ComR$) and total energy used for all communication steps (or $ComW$). Several ways of using these measures for communication analysis were also proposed, e.g. determining the sets computed given these measures, using these measures as restrictions imposed on computations and determining problems decided given such measures, also explored in [7, 5, 4].

In this paper, we explore the set of numbers that can be computed with (un)bounded dynamical communication measures. First, we improve a previous result on computational completeness of ECPe systems. In [1], and in a follow-up work in [8], ECPe systems having four membranes were shown to be computationally complete. In this paper, we show computational completeness using only two membranes by simulating a matrix grammar with appearance checking. We also address a conjecture given in [1] stating that the set of numbers computed with bounded $ComX$, $X \in \{N, R, W\}$ are restricted to semilinear sets. We first show that only semilinear sets can be computed with bounded $ComW$ if the output region only acts as a receiver. We then show a class of non-semilinear sets that can be computed with only two membranes and bounded communication steps.

The paper is organized as follows: we first discuss some preliminaries, including the formal definition of ECPe systems in Section 2. We present our main results in Section 3 and provide our conclusions in Section 4.

2 Preliminaries

It is assumed that the readers are familiar with concepts in formal languages [6] and membrane computing [10]. We mention some concepts used throughout this paper.

Definition 1. Matrix Grammar with appearance checking *A matrix grammar with appearance checking is a tuple $G = (N, T, S, M, F)$ where N is a set of non-terminal symbols, T is a set of terminal symbols, $S \in N$ is the start symbol, and M is a set of matrices; each matrix has the form $(A_1 \to x_1, \dots, A_n \to x_n)$ where $n \geq 1$, $A_i \in N$, $x_i \in (N \cup T)^*$ for $1 \leq i \leq n$. The set F is a set of occurrences of rules in the matrices of M, i.e. $A \to x \in F$, if and only if, there is a matrix $m \in M$ that contains the rule $A \to x$.*

Let $m : (A_1 \to x_1, \dots, A_n \to x_n)$ be a matrix in a matrix grammar with appearance checking G. Matrix m derives z from w, denoted by $w \Rightarrow_m z$, if there is a set of strings $w_1, \dots, w_n, w_{n+1}$ such that $w = w_1$, $z = w_{n+1}$, and for each $i = 1, 2, \dots, n$, one of two cases hold: (a) $w_i = w_i' A_i w_i''$ and $w_{i+1} = w_i' x_i w_i''$ (b) $w_i = w_{i+1}$, A_i does not appear in w_i and $A_i \to x_i \in F$. When w results to z through some matrix, we write $w \Rightarrow z$. We write $w \Rightarrow^* z$ to denote the case when w results to z through zero or more applications of matrices in M. The language of G is then $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$.

Definition 2. Matrix Grammar with appearance checking in binary normal form *A matrix grammar with appearance checking in binary normal form is a matrix grammar $G = (N, T, S, M, F)$ where $N = N_1 \cup N_2 \cup \{S, \#\}$, these three sets are mutually disjoint and matrices are in the following forms:*

1. $(S \to XA)$ *where* $X \in N_1$, $A \in N_2$
2. $(X \to Y, A \to x)$, *where* $X, Y \in N_1$, $A \in N_2$, $x \in (N_2 \cup T)^*, |x| \leq 2$
3. $(X \to Y, A \to \#)$ *where* $X, Y \in N_1$, $A \in N_2$,
4. $(X \to \lambda, A \to x)$ *where* $X \in N_1, A \in N_2$, *and* $x \in T^*$, $|x| \leq 2$

There is only one rule of type 1 *and* F *consists of all rules* $A \to \#$ *contained in matrices of type* 3.

The following are known results [10]: First, the length sets of family of languages computed using matrix grammars with appearance checking is equal to the class of recursively enumerable sets of numbers. Also, the language of any matrix grammar with appearance checking can be computed by a matrix grammar with appearance checking in binary normal form.

2.1 ECPe systems

The following formal definition of ECPe systems is based on the definition given in [1]:

Definition 3. ECPe systems *An Evolution-Communication P system with energy (ECPe system) is a construct of the form $\Pi = (O, e, \mu, w_1, \ldots, w_m, R_1, R'_1, \ldots, R_m, R'_m, i_o)$ where m is the total number of membranes; O is the alphabet of objects; $e \notin O$ is a special object, μ is a hierarchical membrane structure, w_h is the initial multiset over O^* in region h $(1 \leq h \leq m)$.*

The set R_h consists of evolution rules for region h; each evolution rule has the form $a \to v$ where $a \in O$, $v \in (O \cup \{e\})^*$. The set R'_h is the set of communication rules for membrane h. There are three types of communication rules: symport-in, symport-out and antiport. A symport rule takes one of the following forms: (ae^i, in) or (ae^i, out), where $a \in O$, $i \geq 1$. An antiport rule takes the form $(ae^i, out; be^j, in)$ where $a, b \in O$ and $i, j \geq 1$. The value $i_o \in \{0, 1, \ldots, m\}$ is the output region. When $i_o = 0$, then the output region is the environment.

In an ECPe system, evolution rules are in the form of multiset-rewriting, however, only one object is located in the left-hand side of the rule (also called *non-cooperative*). Upon application of an evolution rule $a \to v \in R_h$, a copy of a in region h is removed and replaced with a multiset v. Note that while some copies of e can occur in the multiset v, the object a in the left-hand side cannot be e, i.e. this special object cannot evolve. The form of communication rules are adapted from a variant called P systems with symport and antiport [9]. Upon application of a symport-in rule $(ae^i, in) \in R'_h$, i copies of e outside of membrane h are consumed to transport inside region h a copy of a from the outside region. The reverse of such process is performed when applying a symport-out rule $(ae^i, out) \in R'_h$. We say that the *energy* of either symport rule (ae^i, in) or

(ae^i, out) is equal to i. When applying an antiport rule $(ae^i, in; be^j, out) \in R'_h$, aside from a copy of a outside and a copy of b inside membrane h, there should be i copies of e outside and j copies of e inside membrane h. During application, a and b swap position, and the copies of e used are consumed. We say that the *energy* of antiport rule $(ae^i, in; be^j, out)$ is equal to $i + j$.

In this paper, we focus on ECPe systems that apply rules in the same manner as most P systems, i.e. rules are applied in a nondeterministic and maximally parallel manner. We say that a configuration is a state of an ECPe system, consisting of the multiset at each region. A transition from a configuration C to another configuration C' through a maximally parallel application of rules is denoted by $C \Rightarrow C'$; a sequence of zero or more transitions from a configuration C to another configuration C' is denoted by $C \Rightarrow^* C'$. The configuration of an ECPe system at a time unit i is denoted by C_i ($i \geq 0$). An ECPe system computes by starting from the initial configuration C_0 to a halting configuration C_h such that $C_0 \Rightarrow^* C_h$. We say that $C_0 \Rightarrow^* C_h$ is a *successful computation.* A halting configuration C_h is a configuration where no more rules can be applied to any copies of objects in each region.

A set of numbers can be computed by an ECPe system as follows: the natural number produced as output by a successful computation is the number of objects (including count of e's) in the output region of the halting configuration; the set of numbers generated in this way is exactly the set of numbers computed by an ECPe system. We shall denote this set by $N(\Pi)$.

2.2 Dynamical Communication Measures for ECPe systems

The dynamical communication complexity parameters introduced in [1] are initially determined at the transition level:

$$\begin{aligned} ComN(C_i \Rightarrow C_{i+1}) &= \begin{cases} 1 \text{ if at least one communication rule is used} \\ \quad \text{in the transition } C_i \Rightarrow C_{i+1}, \\ 0 \text{ otherwise} \end{cases} \\ ComR(C_i \Rightarrow C_{i+1}) &= \text{the number of communication rules} \\ &\quad\ \text{used in the transition } C_i \Rightarrow C_{i+1} \\ ComW(C_i \Rightarrow C_{i+1}) &= \text{the total energy considering all applications} \\ &\quad\ \text{of communication rules used} \\ &\quad\ \text{in the transition } C_i \Rightarrow C_{i+1} \end{aligned}$$

These parameters are related as follows: $ComN \leq ComR \leq ComW$. Let $X \in \{N, R, W\}$:

$$\begin{aligned} ComX(\delta) &= \textstyle\sum_{i=0}^{h-1} ComX(C_i \Rightarrow C_{i+1}) \text{ where } \delta : C_0 \Rightarrow C_1 \Rightarrow \\ &\quad \ldots \Rightarrow C_h \text{ is a halting computation,} \\ ComX(n, \Pi) &= \min\{ComX(\delta) \mid \ \delta : C_0 \Rightarrow C_1 \Rightarrow \ldots \Rightarrow C_h \text{ in} \\ &\quad \Pi \text{ with the result } n\}, \\ ComX(\Pi) &= \max\{ComX(n, \Pi) \mid n \in N(\Pi)\} \\ ComX(Q) &= \min\{ComX(\Pi) \mid Q = N(\Pi)\} \end{aligned}$$

We let $NFComX_m(k, sym_p, anti_q)$ be the class of set of numbers computed by ECPe systems Π with at most $m \geq 1$ membranes, $ComX(\Pi) \leq k$, symport rules of maximal energy at most $p \geq 0$ and antiport rules of maximal energy at most $q \geq 0$. When one of the parameters m, k, p, q is not bounded, we replace the respective value by $*$. When $k = 0$, we simply omit sym_0 and $anti_0$. When $m = 1$, since it is obvious that no antiport rules can be applied, we simply omit $anti_0$.

In this study, we also explore ECPe systems with a particular restriction on output, i.e. when the output region is the environment ($out = env$), and when the output region only functions as a receiving region ($out = rec$). The class of set of numbers computed with such a restriction is represented by the following notation: $NFComX_m(k, sym_p, anti_q, out = \alpha)$ where $\alpha \in \{rec, env\}$. When $m = 2$ and $out = rec$, we simply omit $anti_0$.

The inclusions below directly follow from the definitions (as stated in [1]).

Lemma 1. ([1]) $NFComX_m(k, sym_p, anti_q) \subseteq NFComX_{m'}(k', sym_{p'}, anti_{q'}) \subseteq NRE$ for $X \in \{N, R, W\}$ and for all $1 \leq m \leq m'$, $0 \leq p \leq p'$, $0 \leq q \leq q'$; each of m', p', q' can also be equal to $*$.

From [10] (and mentioned in [1]), the class of set of numbers computed by an ECPe system with no communication is the same as the class of semilinear sets.

Theorem 1. ([10]) $NFComX_*(0) = NFComX_1(0) = SLIN$ for $X \in \{N, R, W\}$, $m \geq 1$.

The system considered in [10] is a one-membrane Transition P system where the output region is the skin membrane. The set of numbers computed in a single membrane system with no communication and counting objects in the skin, like the systems in Theorem 1, is exactly the class of semilinear sets. When assigning the environment as output region, the output numbers are determined from the set of communication rules. Thus, the following observation is true about bounded $ComW$.

Fact 1 *$NFComW_1(k, sym_*, out = env) \subset NFIN$ for $k \geq 0$.*

We now observe that when communication is unbounded, ECPe systems having environment as output holder can compute semilinear sets. We use the relation in Theorem 1.

Fact 2 *$NFComW_1(*, sym_p, out = env) \supseteq SLIN$ for $p \geq 1$.*

Proof. To prove this, we let Π be a one-membrane ECPe system without communication rules. We then show that there is a one-membrane ECPe system $\bar{\Pi}$ having environment as output region and $N(\Pi) = N(\bar{\Pi})$.

Given $\Pi = (O, e, [_1]_1, w_1, R_1, \emptyset, 1)$, we construct $\bar{\Pi} = (\bar{O}, e, [_1]_1, w_1, \bar{R}_1, \bar{R}'_1, 0)$ as follows: first, we replace all occurrences of e in Π with e'. We determine the set $O' = \{\alpha \mid \alpha \in O \cup \{e'\}$ and $\nexists r : \alpha \to v,\ v \in (O \cup \{e'\})^*\}$. The set O' is the set of all objects not used in the left hand side of any rule, thus, $N(\Pi)$ only considers the count of objects in O'. For every $\alpha \in O'$, the following rules are added: (a) $\alpha \to \bar{\alpha}e \in \bar{R}_1$ (b) $(\bar{\alpha}e, out) \in \bar{R}'_1$. This provides $\bar{O} = O \cup \{\bar{\alpha} \mid \alpha \in O'\}$. □

Whether only semilinear sets can be generated by one-membrane ECPe systems with unbounded communication, i.e. $NFComW_1(*, sym_*, out = env) = SLIN$, is yet to be proven.

We now recall the results about the computing power of ECPe systems having unbounded communication.

Theorem 2. ([1, 8]) For $X \in \{N, R, W\}$:

- $NFComX_m(*, sym_p, anti_q) = NRE$ for $m \geq 4$, $p \geq 2$, $q \geq 0$.
- $NFComX_m(*, sym_p, anti_q) = NRE$ for $m \geq 4$, $p \geq 1$, $q \geq 2$.

As can be observed, the previous result shows that four membranes suffice to show computational completeness. When only symport rules are allowed, the maximum energy used in a rule is at most two, while when incorporating antiport rules the maximum energy for both symport and antiport rules are the lowest possible values.

Step	Region 1	Membrane 2	Region 2
1			$m_i \to m'_{i1}m''_{i1}e$
2		$(m'_{i1}e, out)$	$m''_{i1} \to m'''_{i1}$
3	$m'_{i1} \to e$		$m'''_{i1} \to m^v_{i1}m'_{i2}e$
4		$(Xe, in; m^v_{i1}e, out)$	$m'_{i2} \to m''_{i2}$
5	$m^v_{i1} \to e$		$X \to e$ $m''_{i2} \to m^v_{i2}$
6		$(Ae, in; m^v_{i2}e, out)$	
7	$m^v_{i2} \to m^s_i \alpha e$ where $\alpha = Yx$ if type 2 or $\alpha = x$ if type 4		$A \to \lambda$
8		$(m^s_i e, in)$	
9			$m^s_i \to m_{ch}$

Table 1. Simulating a type 2 or 4 rule

Step	Region 1	Membrane 2	Region 2
1			$m_i \to m'_{i1}m''_{i1}e$
2		$(m'_{i1}e, out)$	$m''_{i1} \to m'''_{i1}$
3	$m'_{i1} \to e$		$m'''_{i1} \to m^v_{i1}m'_{i2}e$
4		$(Xe, in; m^v_{i1}e, out)$	$m'_{i2} \to m''_{i2}$
5	$m^v_{i1} \to e$		$X \to e$ $m''_{i2} \to m^{\bar{a}_1}_{i2}m^{\bar{a}_2}_{i2}e$
6		$(m^{\bar{a}_2}_{i2}e, out)$	
7	$m^{\bar{a}_2}_{i2} \to m^s_i Y$		
8		$(m^s_i e, in; m^{\bar{a}_1}_{i2}e, out)$	
9	$m^{\bar{a}_1}_{i2} \to \lambda$		$m^s_i \to m_{ch}$

Table 2. Simulating a type 3 rule where a nonterminal symbol A does not occur in the current derivation

3 Main Results

3.1 On ECPe systems with Unbounded Communication

We now show that we can generate the class of recursively enumerable sets of numbers with two membranes and unbounded communication.

Theorem 3. *Let $G = (N, T, S, M, F)$ be a matrix grammar with appearance checking. There is a two-membrane ECPe system Π that computes the length-set of $L(G)$.*

Proof. We shall use a matrix grammar in binary normal form given in Definition 2 for our proof. We impose a total order on the matrices in M so that we can label each matrix from 1 to $|M|$ and uniquely label each matrix as m_i where $1 \leq i \leq |M|$. We construct an ECPe system Π as follows:

$$\Pi = (O, e, [_1[_2]_2]_1, w_1, w_2, R_1, \emptyset, R_2, R'_2, 1)$$

where $O = N \cup T \cup \{m_{ch}, \#\} \cup \{m_i, m^s_i, m^v_{ij}, m'_{ij}, m''_{ij}, m'''_{ij}, m^{\bar{a}_1}_{i2}, m^{\bar{a}_2}_{i2}, m^{\bar{a}_3}_{i2} \mid 1 \leq i \leq |M|, 1 \leq j \leq 2\}$, $w_1 = XA$ where $(S \to XA) \in M$ and $w_2 = m_{ch}$. The sets R_1, R'_2 and R_2 of rules include the rules given in Tables 1 and 2, as well as the following rules:

for region 1:	for membrane 2:	for region 2:	
$m^s_j \to \#$	$(Ae, in; m^{\bar{a}_1}_{j2}e, out)$	$m_{ch} \to m_i$	$m^v_{k2} \to \#$
$\# \to \#$		$m^v_{i1} \to \#$	$\# \to \#$

where $1 \leq i \leq |M|$, m_j $(1 \leq j \leq |M|)$ is a type 3 matrix in M and m_k $(1 \leq k \leq |M|)$ is a type 2 or type 4 matrix in M. Computation of Π proceeds as follows:

Initially, region 1 contains two nonterminal symbols, X and A where the matrix $(S \to XA)$ occurs in M. This simulates the application of the first matrix in G. Region 2 contains an object m_{ch}. The system Π nondeterministically chooses a matrix in M that can be applied to the current derivation given in region 1. Shown in Table 1 is the step-by-step computation that simulates a type 2 or a type 4 rule, whereas Table 2 shows computations simulating a type 3 rule.

Suppose in a transition, a rule $m_{ch} \to m_i$ is in region 2 and m_i is a type 2 matrix where $m_i : (X \to Y, A \to x)$. In order for m_i to be correctly simulated, symbols X and A must be present in region 1. Steps 2 to 4 in Table 1 are used to validate and remove the occurrence of the symbol X in region 2 while steps 5 to 6 in the table validate and remove the occurrence A. The occurrence of the object m^s_i in region 2 at step 8 indicates a successful validation. The rule $m^s_i \to m_{ch}$ signals completion of simulating m_i. Shown below is the computation for a successful simulation of a type 2 matrix $m_i : (X \to Y, A \to x)$:

Step 0: $[_1\ XA\ [_2\ m_i]_2]_1$	Step 5: $[_1\ eA\ [_2\ em^v_{i2}]_2]_1$
Step 1: $[_1\ XA\ [_2\ m'_{i1}m''_{i1}e]_2]_1$	Step 6: $[_1\ m^v_{i2}\ [_2\ A]_2]_1$
Step 2: $[_1\ XAm'_{i1}\ [_2\ m'''_{i1}]_2]_1$	Step 7: $[_1\ m^s_iYxe\ [_2\ ]_2]_1$
Step 3: $[_1\ XAe\ [_2\ m^v_{i1}m'_{i2}e]_2]_1$	Step 8: $[_1\ Yx\ [_2\ m^s_i]_2]_1$
Step 4: $[_1\ m^v_{i1}A\ [_2\ Xm''_{i2}]_2]_1$	Step 9: $[_1\ Yx\ [_2\ m_{ch}]_2]_1$

If m_i is a type 4 matrix $m_i : (X \to \lambda, A \to x)$, then Yx in steps 7 to 9 is replaced with x. Note that in cases where either X or A doesn't occur in region 1, the antiport rules $(Xe, in; m^v_{i1}e, out)$ or $(Ae, in; m^v_{i2}e, out)$, respectively, cannot be executed. Instead, rule $m^v_{i1} \to \#$ is executed in the former case while $m^v_{i2} \to \#$ is executed in the latter case. In both cases, the trap symbol # occurs in region 1. This signals a non-halting computation due to the evolution rule $\# \to \#$.

Suppose in a transition, a rule $m_{ch} \to m_i$ is in region 2 and m_i is a type 3 matrix where $m_i : (X \to Y, A \to \#)$. Since m_i is of type 3, the occurrence of both X and A in the current derivation leads to an unacceptable string when m_i is applied. In such case, A becomes # and any succeeding derivations will produce a non-acceptable string. When only X occurs and A does not exist in the current derivation, rule m_i is applied such that X becomes Y. Table 2 shows the sequence of rules that simulates matrix m_i when only X exists in the current derivation (i.e. when only X exists in region 1). Shown below is the computation for such successful simulation:

Step 0: $[_1\ X\ [_2\ m_i]_2]_1$
Step 1: $[_1\ X\ [_2\ m'_{i1}m''_{i1}e]_2]_1$
Step 2: $[_1\ Xm'_{i1}\ [_2\ m'''_{i1}]_2]_1$
Step 3: $[_1\ Xe\ [_2\ m^v_{i1}m'_{i2}e]_2]_1$
Step 4: $[_1\ m^v_{i1}\ [_2\ Xm''_{i2}]_2]_1$
Step 5: $[_1\ e\ [_2\ em^{\bar{a}_1}_{i2}m^{\bar{a}_2}_{i2}e]_2]_1$
Step 6: $[_1\ em^{\bar{a}_2}_{i2}\ [_2\ em^{\bar{a}_1}_{i2}]_2]_1$
Step 7: $[_1\ em^s_iY\ [_2\ em^{\bar{a}_1}_{i2}]_2]_1$
Step 8: $[_1\ m^{\bar{a}_1}_{i2}Y\ [_2\ m^s_i]_2]_1$
Step 9: $[_1\ Y\ [_2\ m_{ch}]_2]_1$

As can be observed, similar to simulating a type 2 or type 4 matrix, absence of X in region 2 leads to application of rule $m^v_{i1} \to \#$ in membrane 2 leading to a non-halting computation. In the case where both X and A occurs in region 2, the rule $(Ae, in; m^{\bar{a}_1}_{i2}e, out)$ is applied simultaneously with $(m^{\bar{a}_2}_{i2}e, out)$ in step 5 so that in step 6, both $m^{\bar{a}_1}_{i2}$ and $m^{\bar{a}_2}_{i2}$ are in region 1. This will prevent the application of the antiport rule $(m^s_ie, in; m^{\bar{a}_1}_{i2}e, out)$ in step 7. Instead, the evolution rule $m^s_i \to \#$ is applied, leading to a non-halting computation.

Corollary 1. *$NFComX_m(*, sym_p, anti_q) = NRE$ for $X \in \{N, R, W\}$, $m \geq 2$, $p \geq 1$, $q \geq 2$.*

Proof. Follows from Theorem 3.

3.2 On ECPe systems with Bounded *ComW*

In this section, we consider systems where every communication always uses the output region as receiver.

Lemma 2. *$NFComW_2(k, sym_*, out = rec) = SLIN$ for $k \geq 0$.*

Proof. Let Π be a two-membrane ECPe system where $ComW(\Pi) = k$ and the output region is only a receiving region. We show that there is a one-membrane ECPe system $\bar{\Pi}$ having $ComW(\bar{\Pi}) = 0$ and $N(\Pi) = N(\bar{\Pi})$.

If the output region of Π is the environment, then this is true via Fact 1. Without loss of generality, let us assume the output region is the skin. Let

$\Pi = (O, e, [_1[_2]_2]_1, w_1, w_2, R_1, R'_1, R_2, R'_2, 1)$. Suppose only one object is communicated so that $ComW(\Pi) = 1$. This also implies that only one communication rule is applied in a computation. Let this rule be $r : (ae, out) \in R'_2$, $a \in O$. Since evolution rules are non-cooperative, the communicated object a will be subjected to the same rules (and thus, production of the same multiset) regardless of the time it was communicated in the output region. If another computation exists such that rule r is used at a later or earlier time, the output region in the halting configurations of the two computation paths contain the same multiset. This same multiset is also obtained in the halting configuration when a is in the initial multiset of the output region. This means, $\bar{\Pi}$ can be generated having only the rules for the output region and including the object a in the initial multiset.

For cases where there is a halting computation path in Π without application of any communication rule, a rule $a \to \lambda$ can be included. For cases where there are several halting computation paths having distinct communication rules applied (e.g. (b_1e, out) in one computation path, (b_2e, out) in another, until (b_je, out) in the j^{th} computation path), we can set an initial symbol, say α, in $\bar{\Pi}$. We then include the rules $\alpha \to b_1$, $\alpha \to b_2$ and $\alpha \to b_j$ in $\bar{\Pi}$.

To extend this to the case where $ComW(\Pi) = k$, we shall follow the same technique as discussed above. We construct a one-membrane ECPe system $\bar{\Pi}$ having the following characteristics: (a) Rules of its skin contain the rules in the output region of Π and (b) the initial multiset of its skin is composed of w_1 and an inital symbol α. Let V be the set of multisets communicated to the skin for all halting computations of Π. We add $|V|$ production rules $\alpha \to v$ where $v \in V$.

The set V can be obtained as follows: Suppose $R'_2 = \{r_1 : (a_1e^{p_1}, out), r_2 : (a_2e^{p_2}, out), \ldots,\ r_n : (a_ne^{p_n}, out)\}$. For each halting computation path δ, let $App(\delta) = (c_1, c_2, \ldots,\ c_n)$ where each value c_i refers to the total number of applications of rule r_i in the halting computation δ. Since $ComW(\Pi) = k$, the total energy used for communication in a halting computation is at most k. Thus, $App(\delta) = (c_1, c_2, \ldots,\ c_n)$ must follow the constraint:

$$0 \leq c_1 + c_2 + \ldots + c_n \leq k \qquad (1)$$

Let $Q = \{(c_1, c_2, \ldots, c_n) \mid (c_1, c_2, \ldots, c_n) \text{ satisfies the Inequality (1)}\}$. Also, let $Q_v = \{v = a_1^{c_1} a_2^{c_2} \ldots a_n^{c_n} \mid (c_1, c_2, \ldots, c_n) \in Q\}$. Then, set V is a subset of Q_v ($V \subseteq Q_v$). □

This result can be extended to handle multiple membranes. In such a case, all neighboring regions will be considered in determining the set V of possible multisets communicated to the output region. Thus, the following theorem is given.

Theorem 4. *$NFComW_*(k, sym_*, anti_*, out = rec) = SLIN$ for $k \geq 0$.*

Let Π be an ECPe system having the following characteristics: (a) only symport rules are used, (b) $ComW(\Pi) = k$ and (c) communicated objects are not used in their respective receiving regions. The set V of all possible multisets communicated to the output region in Π can be obtained as a subset of multisets obtained via an inequality similar to Inequality (1). Since any multiset $v \in V$ is not influenced by the multiset communicated from the output region (and vice versa), we can use the same technique of Lemma 2 to construct a one-membrane ECPe system $\bar{\Pi}$ having the following characteristics: (a) Evolution rules of the skin of $\bar{\Pi}$ contain the evolution rules in the output region of Π and (b) the initial multiset of the skin of $\bar{\Pi}$ is composed of the initial multiset in the output region as well as a symbol α. We add $|V|$ production rules $\alpha \to v$ where $v \in V$ and R'_1 of $\bar{\Pi}$ is constructed in the following way: for all (ae^p, tar), $p \geq 1$ and $tar \in \{in, out\}$, having the output region as the sending region, $(ae^p, out) \in R'_1$. Thus, the following corollary is given.

Corollary 2. *Let Π be an m-membrane ECPe system where $ComW(\Pi) = k$, only symport rules are used ($m \geq 2$, $k \geq 0$) and all communicated objects are not used in their respective receiving regions. There is a one-membrane ECPe system $\bar{\Pi}$ having $N(\Pi) = N(\bar{\Pi})$ and $ComW(\bar{\Pi}) \leq k$.*

Corollary 2 holds as long as the succeeding evolutions from the communicated objects do not produce e's or objects that can possibly be communicated in the succeeding transitions.

3.3 On ECPe systems with Bounded $ComX$, $X \in \{N, R\}$

The example below shows a non-semilinear set that can be computed with bounded communication steps.

Example 1.

$$\bar{\Pi} = (O, e, [_1[_2]_2]_1, w_1, w_2, R_1, \emptyset, R_2, R'_2, 1)$$

where $O = \{a', a, b, \alpha, \beta, \theta\} \cup \{c_i \mid 1 \leq i \leq 5\}$, $R_1 = \{a' \to ae, a \to c_1, c_5 \to aaee, b \to \beta, \theta \to \alpha\} \cup \{c_i \to c_{i+1} \mid 1 \leq i \leq 4\}$, $R_2 = \{a \to be, \beta \to \theta e, \alpha \to \alpha\}$, and $R'_2 = \{(ae, in), (be, out), (\beta e, in), (\theta e, out), (\alpha e, in)\}$.

Computation of Π proceeds as follows: Initially, the objects a' in region 1 evolve via rules $a' \to ae$.

These rules produce a copy of a and e in region 1. At time $t = 1$, the system nondeterministically chooses between application of rules $a \to c_1$ or applying the symport rule (ae, in). In the latter case, the e's produced from the previous rule will be consumed to transfer object a in region 2. In the next three time steps, the system reaches a halting configuration via consecutive use of rules $a \to be$, (be, out) and $b \to \beta$. Such computation produces the number 1 since only the object β is present in region 1.

When the symport rule is not used at time $t = 1$, the computation proceeds by continuously applying the sequence, $a \to c_1$, $c_i \to c_{i+1}$ $(1 \leq i \leq 4)$ and $c_5 \to aaee$ in region 1. At step $1 + 6n$, $n \geq 0$, there are 2^n copies of a and $2^{n+1} - 1$ copies of e in region 1. A successful computation applies the symport rule (ae, in) to all copies of a so that in the next step, there are $2^n - 1$ copies of e in region 1 and 2^n copies of a in region 2. In the next two time steps, the copies of a in region 2 evolve via consecutive use of rules $a \to be$ and (be, in) so that there will be an additional 2^n copies of b in region 1.

The copies of b in region 1 will evolve via the rule $b \to \beta$. Since there are only $2^n - 1$ copies of e in region 1, there will only be $2^n - 1$ applications of rule $(\beta e, in)$ in the succeeding step. After such applications, there will only be one copy of β in region 1 and $2^n - 1$ copies of β in region 2. The next two steps proceeds as follows: the copies of β in region 2 evolve via rule $\beta \to \theta e$ and the θs are communicated to region 1 via rule $(\theta e, out)$. The copies of θ in region 1 evolve via rule $\theta \to \alpha$. At this point, no more rules are applicable, leading the system to a halting state. Region 1 has $2^n - 1$ copies of α and one copy of β, thus the system outputs 2^n.

We now look at the case where copies of a in region 1 are not communicated at the same time. Let step $1+6n$ be the step where the first symport rule (ae, in) is applied. Also, let us say there are only $k < 2^n$ applications of this symport rule. The next sequence of transitions proceeds as follows:

1. At step $1+6n+3$, there are k copies of b $(k < 2^n)$ and the remaining copies of e are at least 2^n in region 1.
2. At step $1+6n+4$, each b produces β.
3. At step $1+6n+5$, since there are at least 2^n copies of e and less than 2^n copies of β, there are some e's left in region 1 after the application of rule $(\beta e, in)$.
4. At step $1+7n$, rule $\beta \to \theta e$ is applied in region 2 and at step $1+7n+1$, the rule $(\theta e, out)$ is applied in membrane 2. At step $1+7n+2$, the rule $\theta \to \alpha$ is applied in region 1.
5. At step $1+7n+3$, since there are some e's left in region 1, the communication rule $(\alpha e, in)$ is applied, leading the system to a non-halting computation due to the rule $\alpha \to \alpha$ in region 2.

Note that since not all copies of a are communicated at step $1+6n$, at step $1+7n$ there are some copies of a and additional copies of e in region 1. If some of these copies trigger the rule (ae, in) in the next step, application of rule $(\beta e, in)$ happens at step $1+7n+5$, which occurs after item 5.

The description of the above computation shows that $N(\bar{\Pi}) = \{2^n \mid n \geq 0\}$. For each successful computation, exactly four communication steps were used, and in each communication step, only one communication rule is used. Thus, $ComN(\bar{\Pi}) = ComR(\bar{\Pi}) = 4$.

The example above leads to the following theorem.

Theorem 5. *$NFComX_m(k, sym_p, anti_q) - SLIN \neq \emptyset$ for $X \in \{N, R\}$, $m \geq 2$, $k \geq 4$, $p \geq 1$, $q \geq 0$.*

In the next theorem, we extend the construction in Example 1 to generate any set composed of powers of j.

Theorem 6. *For $N_j = \{j^n \mid n \geq 0\}$, $j > 1$:*

1. *$ComN(N_j) = ComR(N_j) \leq 4$.*
2. *$N_j \in NFComX_m(k, sym_p, anti_0)$ for $X \in \{N, R\}$, $m \geq 2$, $k \geq 4$, $p \geq j-1$.*

Proof. For every $j > 1$, we construct an ECPe system $\bar{\Pi}_j$:

$$\bar{\Pi}_j = (O, e, [_1[_2]_2]_1, w_1, w_2, R_1, \emptyset, R_2, R_2', 1)$$

where $O = \{a', a, b, \alpha, \beta, \theta\} \cup \{c_i \mid 1 \leq i \leq 5\}$, $R_1 = \{a' \to ae^{j-1}, a \to c_1, c_5 \to a^j e^{j(j-1)}, b \to \beta, \theta \to \alpha\} \cup \{c_i \to c_{i+1} \mid 1 \leq i \leq 4\}$, $R_2 = \{a \to be, \beta \to \theta e, \alpha \to \alpha\}$, and $R_2' = \{(ae^{j-1}, in), (be, out), (\beta e, in), (\theta e, out), (\alpha e, in)\}$. It can be observed that $\bar{\Pi}_j = \{j^n \mid n \geq 0\}$ and $ComN(\Pi) = ComR(\Pi) = 4$.

The idea for how the system computes is similar to the ECPe system given in Example 1 for generating 2^n. At step $1 + 6n$, $n \geq 0$, there are j^n copies of a and $j^{n+1} - 1$ copies of e in region 1. The latter value is computed as

$$\begin{aligned}\sum_{i=0}^{n}(j^i)(j-1) &= (j-1)\sum_{i=0}^{n}(j^i)\\ &= (j-1)\left(\frac{1-j^{n+1}}{1-j}\right)\\ &= j^{n+1} - 1\end{aligned}$$

A successful computation applies the symport rule (ae^{j-1}, in) to all copies of a so that in the next step, there are j^n copies of a in region 2 and $j^n - 1$ copies of e in region 1. The value $j^n - 1$ is computed as $(j^n(j) - 1) - j^n(j-1)$ where the subtrahend is obtained from the copies of e consumed due to applications of rule r'_{21}. The copies of a in region 2 becomes b and gets transported back to region 1 via the rule $a \to be$ and (be, out), respectively. The copies of b in region 1 becomes β via rule r_{18}. Since there are only $j^n - 1$ copies of e in region 1, there will only be $j^n - 1$ applications of rule $(\beta e, in)$ in the succeeding step. This leaves one copy of β in region 1. The copies of β in region 2 becomes θ and gets transported back to region 1 via the rule $\beta \to \theta e$ and $(\theta e, out)$, respectively. Afterwards, no more rules are applicable leaving region 1 with one copy of β and $j^n - 1$ copies of θ.

In the event that the symport rule (ae^{j-1}, in) is not applied to all copies of a in region 1 at the same time, then after the first application of this symport rule, say at time step $1 + 6n$, there will be less than j^n copies of β and at least j^n copies of e at step $1 + 6n + 5$. As a consequence, not all e's will be used for the rule $(\beta e, in)$. The extra e's will be used in executing at least one application of rule $(\alpha e, in)$ at step $1 + 7n + 3$ leading the system to a non-halting state. □

Although a bounded number of communication steps is enough to compute any set $\{j^n \mid n \geq 0\}$, the ECPe system constructed for Theorem 6 needed a symport rule having energy $j - 1$ to compute j^n. It is interesting to determine whether computing the set $\{j^n \mid n \geq 0\}$ using four communication steps can be done using rules with maximum energy of less than $j - 1$.

The next theorem shows that classes of sets of numbers involving summation of exponential terms with several distinct bases can still be computed with four communication steps. However, the number of communication rules depends on the number of bases. Let $Q \in NFComNR_m(k, k', sym_p, anti_q)$ if and only if $Q \in NFComN_m(k, sym_p, anti_q)$ and $Q \in NFComR_m(k', sym_p, anti_q)$.

Theorem 7. *Let $N \in SLIN$ and $Q = \left\{ \sum_{t=1}^{s} j_t^n + \alpha \mid n \geq 0,\ j_t > 1, \alpha \in N \right\}$. Then $Q \in NFComNR_m(k, k', sym_p, anti_q)$ for $m \geq 2$, $k \geq 4$, $k' \geq s + 3$, $p \geq j - 1$, $q \geq 0$, $j = \max\{j_1, \ldots, j_s\}$.*

Proof. We construct a 2-membrane ECPe system Π generating Q as follows: first, since $N \in SLIN$, there is a set of evolution rules that can be defined in the output region in order to produce α objects in a halting configuration. For each j_t^n, we shall use multisets and rules similar to those given in Example 1. Also, for each object a, a', c_1 to c_5, we append a subscript t e.g. a becomes a_t. In the first communication step of a successful computation, rules $(a_i e^{j-1}, in)$ for $1 \leq i \leq s$ are used. In the next three communication steps, rule (be, out), $(\beta e, in)$ and $(\theta e, out)$ are used, respectively. □

3.4 The Power of Including Antiport Rules

Notice that we use symport rules only in the ECPe system constructions described in the previous section. In the next theorem, we reduce the values of

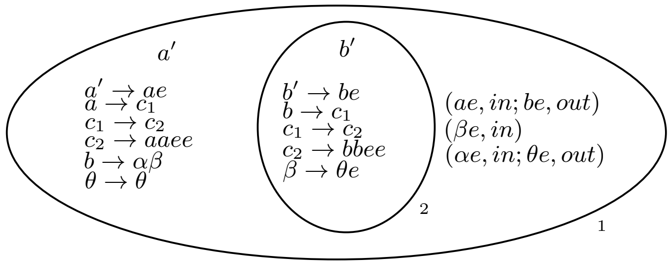


Fig. 1. An ECPe system generating $\{2^n + 1 \mid n \geq 0\}$. The output region is the skin.

$ComN$ and $ComR$ in Theorem 6 (from four to two) by including an antiport rule.

Theorem 8. *For $N_j = \{j^n + 1 \mid n \geq 0\}$, $j > 1$:*

1. *$ComN(N_j) = ComR(N_j) \leq 2$.*
2. *$N_j \in NFComX_m(k, sym_p, anti_q)$ for $X \in \{N, R\}$, $m \geq 2$, $k \geq 2$, $p \geq 1$, $q \geq j$.*

Proof. For this theorem, we construct an ECPe system Π_j generating $N_j = \{j^n + 1 \mid n \geq 0\}$ for $j > 1$ as follows: $\Pi_j = (O, e, [_1[_2]_2]_1, a', b', R_1, \emptyset, R_2, R'_2, 1)$ where $O = \{a', b', a, b, c_1, c_2, \alpha, \beta, \theta\}$, $R_1 = \{a' \to ae^{j-1}, a \to c_1, c_1 \to c_2, c_2 \to a^j e^{j(j-1)}, b \to \alpha\beta, \theta \to \theta\}$, $R_2 = \{b' \to be, b \to c_1, c_1 \to c_2, c_2 \to b^j e^j, \beta \to \theta e\}$, and $R'_2 = \{(ae^{j-1}, in; be, out), (\beta e, in), (\alpha e, in; \theta e, out)\}$. Fig. 1 shows the description of an ECPe system Π_j when $j = 2$.

Computation for Π_j with $j > 1$ proceeds as follows: At step $1 + 3n$, $n \geq 0$, there are j^n copies of a and $j^{n+1} - 1$ copies of e in region 1. In region 2, there are j^n copies of b and at least j^n copies of e. A successful computation applies the antiport rule $(ae^{j-1}, in; be, out)$ to all copies of a so that in the next step, there are j^n copies of b and $j^n - 1$ copies of e in region 1. (The a's transported in region 2 as well as the remaining e's in the region will no longer be used in the next steps). The copies of b in region 1 will evolve via the rule $b \to \alpha\beta$. Since there are only $j^n - 1$ copies of e in region 1, there will only be $j^n - 1$ applications of rule $(\beta e, in)$ in the succeeding step. The remaining copies of objects in region 1 will then be j^n copies of α and a copy of β. After the production of copies of θ via rule $\beta \to \theta e$, no more rules are applicable in the next step.

In the event that antiport rule $(ae^{j-1}, in; be, out)$ is not applied to all copies of a in region 1 at the same time, then after the first application of the antiport rule, say at time step $1 + 3n$, there will be less than j^n copies of β at step $1 + 4n$ and at least j^n copies of e. The extra e's will be used in executing at least one application of rule $(\alpha e, in; \theta e, out)$ leading the system to a non-halting state. □

The idea for how ECPe systems for Theorem 8 compute is similar to the ECPe systems given in Theorem 6. Combining separate symport rules (ae, in) and (be, out) into one antiport rule $(ae, in; be, out)$ results to a reduction of steps from four to two. However, the resulting output of the system is increased by one (generating $j^n + 1$ instead of j^n). Note also that the construction for Theorem 8 requires an antiport rule having energy j to compute $j^n + 1$ in just two communication steps.

The proof for the next theorem is similar to the proof given in Theorem 7.

Theorem 9. *Let $N \in SLIN$ and $Q = \left\{ \sum_{t=1}^{s} (j_t^n + 1) + \alpha \mid n \geq 0,\ j_t > 1, \alpha \in N \right\}$. Then $Q \in NFComNR_m(k, k', sym_p, anti_q)$ for $m \geq 2$, $k \geq 2$, $k' \geq s+1$, $p \geq 1$, $q \geq j$, $j = \max\{j_1, \ldots, j_s\}$.*

4 Summary

We analyze the computing power of ECPe systems with (un)bounded dynamical communication measures $ComX$, where $X \in \{N, R, W\}$. We provide insights on the class of numbers computed for one-membrane ECPe systems where the environment is set as output region. When using bounded $ComW$, only a finite set of numbers can be computed. When using unbounded $ComW$, semilinear sets of numbers can be computed. Whether only semilinear sets of numbers can be generated by such systems remains an open problem. Also, the class of sets of numbers computed using one-membrane ECPe systems that use skin as output region and (un)bounded non-zero communication has not been addressed.

Our result about computational completeness using only two membranes is an improvement from the results given in [1, 8] that use four membranes. The resources used are almost similar to universality proofs in ECP systems, e.g. as presented in [2]. We now ask the following question: can we construct computationally complete two-membrane ECPe systems that only make use of symport rules? We know that a four-membrane ECPe system is computationally complete even using only symport rules (as presented in [1]). Note, however, that the energy required in the rules are not optimal.

We presented result on ECPe systems having bounded $ComW$ and where the output region only acts as a receiving end of communication. We have shown that:

$$SLIN = NFComW_*(0) = NFComW_*(k, sym_*, anti_*, out = rec)$$

We also extended such result to two-way communication with bounded $ComW$ where the communicated objects are not used in their receiving regions. Any set of numbers computed by such systems can also be computed by a one-membrane ECPe system.

Shown below is a hierarchy of relations considering our results and the results given in [1] and [8]. Let $X \in \{N, R\}$,

$$\begin{gathered} SLIN = NFComX_*(0) \subset NFComX_2(4, sym_1, anti_0) \subseteq \\ NFComX_4(*, sym_2, anti_0) = NRE \\ SLIN = NFComX_*(0) \subset NFComX_2(2, sym_1, anti_2) \subseteq \\ NFComX_2(*, sym_1, anti_2) = NRE \end{gathered}$$

Contrary to a previous conjecture (in [1]), this shows that the class of numbers computed with bounded $ComX$ is strictly greater than the numbers computed with no communication. The relation between no communication and only one $ComX$ remains an open problem. It is also interesting to determine the exact class of sets of numbers computed with $ComX$ of two and $ComX$ greater than two.

In showing the power of bounded communication, we are able to explore a class of set of numbers computed with increasing maximum energy cost, as shown below. Let $j > 1$ and $X \in \{N, R\}$:

$$\begin{gathered} \{j^n \mid n \geq 0\} \in NFComX_2(4, sym_{j-1}, anti_0) \\ \{j^n + 1 \mid n \geq 0\} \in NFComX_2(2, sym_1, anti_j) \end{gathered}$$

It is interesting to determine if we can reduce the cost of the energy used in the rules when computing these sets with two membranes and $ComX$ having the same bounded values. Specifically, is $\{j^n \mid n \geq 0\} \notin NFComX_2(4, sym_{j-2}, anti_0)$? Is $\{j^n + 1 \mid n \geq 0\} \notin NFComX_2(2, sym_1, anti_{j-1})$? Can computing powers of j, $j > 1$ be done using only one communication step?

5 Acknowledgements

R. Juayong would like to thank the DOST-ERDT Scholarship Program, OVPAA-OIL and the College of Engineering, UP Diliman. N. Hernandez is supported by the Vea Family Technology for All Centennial professorial chair. F. Cabarle is grateful for a Faculty Incentive and Research Award (2016-2017) from the College of Engineering, UP Diliman, the PhDIA Project No. 161606 from the UP Diliman, OVCRD, an RLC grant 2016–2017 also from OVCRD and the Soy and Tess Medina Professorial Chair Award for 2016–2017. H. Adorna is supported by the Semirara Mining Corp. professorial chair, UP Diliman.

References

1. Adorna, H.N., Păun, G., Pérez-Jiménez, M.J.: On Communication Complexity in Evolution-Communication P Systems. Romanian Journal of Information Science and Technology 13(2), 113–130 (2010)
2. Alhazov, A.: Communication in Membrane Systems with Symbol Objects. Ph.D. thesis, Universitat Rovira I Virgili (2006)
3. Cavaliere, M.: Evolution–Communication P Systems. In: Păun, G., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing: International Workshop, WMC-CdeA 2002 Curtea de Arges, Romania, August 19–23, 2002 Revised Papers, pp. 134–145. Springer Berlin Heidelberg (2003)
4. Donor, B., Juayong, R.A.B., Adorna, H.N.: On the Communication Complexity of Sorting in Evolution-Communication P systems with Energy. In: 12th Philippine Computing Science Congress (PCSC2012), Canlubang, Laguna. pp. 15–25 (2002)
5. Francia, S.L., Francisco, D.A.A., Juayong, R.A.B., Adorna, H.N.: On Communication Complexity of Some Hard Problems in ECPe Systems with Priority. Philippine Computing Journal 9(2), 14–25 (2014)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1990)
7. Hernandez, N.H.S., Juayong, R.A.B., Adorna, H.N.: On Communication Complexity of Some Hard Problems in ECPe Systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing: 14th International Conference, CMC 2013, Chişinău, Republic of Moldova, August 20-23, 2013, Revised Selected Papers, pp. 206–224. Springer Berlin Heidelberg (2014)
8. Juayong, R.A.B., Adorna, H.N.: A Note on the Universality of EC P Systems with Energy. In: 2nd International Conference on Information Technology Convergence and Services (ITCS), 2010. pp. 1–6 (August 2010)
9. Păun, A., Păun, G.: The Power of Communication: P systems with Symport/Antiport. New Generation Computing 20(3), 295–305 (2002)
10. Păun, G.: Membrane Computing. Springer-Verlag Berlin Heidelberg (2002)
11. Yao, A.C.: Some complexity questions related to distributed computing. In: ACM Symposium on Theory of Computing. pp. 209–213 (1979)

Modelling and Validating an Engineering Application in Kernel P Systems

Raluca Lefticaru[1,2], Mehmet Emin Bakir[3], Savas Konur[1], Mike Stannett[3], Florentin Ipate[2]

[1] School of Electrical Engineering and Computer Science, University of Bradford, West Yorkshire, Bradford BD7 1DP, UK
`{r.lefticaru,s.konur}@bradford.ac.uk`
[2] Department of Computer Science, University of Bucharest, Str. Academiei nr. 14, 010014, Bucharest, Romania
`florentin.ipate@ifsoft.ro`
[3] Department of Computer Science, The University of Sheffield, Regent Court, 211 Portobello, Sheffield S1 4DP, UK
`{mebakir1,m.stannett}@sheffield.ac.uk`

Abstract. This paper illustrates how kernel P systems (or *kP systems*) can be used for modelling and validating an engineering application, in this case a cruise control system for an electric bike. The validity of the system is demonstrated via formal verification, carried out using the kPWorkbench tool. Furthermore, we show how the kernel P system model can be tested using automata and X-machine based techniques.
Keywords: Membrane computing; kernel P systems; cruise control; electric bike; bicycle; verification; testing.

1 Introduction

Nature inspired computational approaches have been on the focus of researchers for several decades. Membrane computing [17] is one of these paradigms that has recently been through significant developments and achievements. For the most up to date results, we refer the reader to [18]. The main computational models are called *P systems*, inspired by the functioning and structure of the living cell.

In recent years, various types or classes of P systems have been introduced and applied to different problems. While these variants provide more flexibility in modelling, this has inevitably resulted in a large pool of P system variants, which do not have a coherent integrating view.

Kernel P (kP) systems have been introduced to unify many variants of P system models, and combine a blend of various P system features and concepts, including (i) complex guards attached to rules, (ii) flexible ways to specify the system structure and dynamically change it and (iii) various execution strategies for rules and compartments.

Kernel P systems are supported by a software suite, called kPWorkbench, which integrates several tools to simulate and verify kP systems models written in a modelling language, called *kP-Lingua*, capable of mapping the kernel P system specification into a machine readable representation.

The usability and efficiency of kP systems have been illustrated by a number of representative case studies, ranging from systems and synthetic biology, e.g. quorum sensing [15], genetic Boolean gates [19] and synthetic pulse generators [1], to some classical computational problems, e.g. sorting [6], broadcasting [10] and subset sum [5].

Here, as an engineering application, we focus on an *e-bike cruise control system.* An *e-bike* (electric bicycle) is a bicycle that uses an integrated rechargeable battery and an electric motor, which provides propulsion. A *cruise control* is an advanced driver-assistance system technology that automatically regulates the speed of a transportation system (such as motor vehicle or electric bicycle) set by the user. From a system design perspective, the validation of the operational safety of any component/feature is very crucial [20]. In this paper, we will model an e-bike cruise control system using kernel P systems and verify its behaviour using the kPWorkbench verification environment. We also show how the kernel P system model can be tested using automata and X-machine based techniques.

This paper is structured as follows: Section 2 introduces the preliminaries and theoretical background. Section 3 presents our modelling approach using kPWorkbench, while Sections 4 and 5 present the verification and testing approach. Finally, conclusions and further work are drawn in Section 6.

2 Background

This section briefly presents cruise control systems, then gives the basic definitions regarding kernel P systems [9], a presentation of the kPWorkbench software suite, and previous testing approaches for membrane systems.

2.1 Cruise control system for an electric bicycle

In this paper, we focus on an e-bike cruise control system. By controlling the speed of the e-bike (or other transportation system), this feature makes the driving experience easier as the user does not have to use the accelerator or brake. For an e-bike system a cruise control feature also assists the user by improving the control of the journey time and controlling the level of exercise undertaken.

From a system design point of view, however, adding a new feature brings in new challenges for the operational safety of the new functionality [20]. Thus validation of additional functionalities of any new technology and their impact on other components of an existing system is important. This will be our focus in this paper.

An e-bike cruise control works as follows (see Figure 1):

- At any time, the system can be at any of the following states:
  (i) pedal bike (Pedal Only – PO, for short)

(ii) pedal bike with power assistance (Pedal Assist – PA)
(iii) maintain constant speed (Cruise Control – CC)
(iv) pedal to charge battery (Pedal Charge – PC)
(v) brake (Brake – BR).

- CC can be activated from PO or PA.
- If CC is cancelled, the system returns to the state from where it was activated i.e., PO or PA, respectively.
- Pedal assist can be requested when the user is pedalling.
- Pedal charge can be requested when the user is pedalling.
- When the user brakes from CC mode, the system returns to PA/PO before going to BR (if brake is still held).
- BR can be reached from PO, PA or PC.
- If the user releases the brake, the system goes to PO, no matter which was the state before Brake[1].

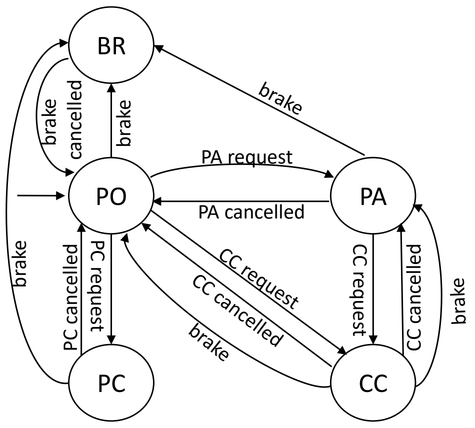


Fig. 1. The state machine representing the behaviour of the e-bike cruise control system.

2.2 Kernel P systems

We first begin recalling the formal definition of kernel P systems (or kP systems).

Definition 1. *A* kP system *of degree n is a tuple $k\Pi = (A, \mu, C_1, \ldots, C_n, i_0)$, where*

- *A is a finite set of elements called objects;*

[1] This happens because from the Brake state, after releasing the brake lever, one can only start to pedal and enter in PO mode. In order to enter in PA mode, the user must first start to pedal and then make a pedal assist request.

- *μ defines the membrane structure, which is a graph, (V, E), where V is a set of vertices representing components (compartments), and E is a set of edges, i. e., links between components;*
- *$C_i = (t_i, w_{i,0})$, $1 \leq i \leq n$, is a compartment of the system consisting of a* compartment type*, t_i, from a set T and an initial multiset, $w_{i,0}$ over A; the type $t_i = (R_i, \rho_i)$ consists of a set of evolution rules, R_i, and an execution strategy, ρ_i;*
- *i_0 is the output compartment where the result is obtained.*

Kernel P systems have features inspired by object-oriented programming: one *compartment type* can have one or more *instances*. These instances share the same set of rules and execution strategies (so will deliver the same functionality), but they may contain different multisets of objects and different neighbours according to the graph relation specified by (V, E). Within the kP systems framework, the following types of evolution rules have been considered so far:

- *rewriting and communication* rule: $x \longrightarrow y\{g\}$, where $x \in A^+$ and y represents a multiset of objects over A^* with potential different compartment type targets (each symbol from the right side of the rule can be sent to a different compartment, specified by its type; if multiple compartments of the same type are linked to the current compartment, then one is randomly chosen to be the target). Unlike cell-like P systems, the targets in kP systems indicate only the types of compartments to which the objects will be sent, not particular instances. Also, for kP systems, complex *guards* can be represented, using multisets over A with relational and Boolean operators [9].
- *structure changing* rules: membrane division, membrane dissolution, link creation and link destruction rules, which all may also incorporate complex guards and that are covered in detail in [9].

In addition to its evolution rules, each compartment type in a kP system has an associated *execution strategy*. The rules corresponding to a compartment can be grouped in blocks, each having one of the following strategies:

- *sequential*: if the current rule is applicable, then it is executed, advancing towards the next rule/block of rules; otherwise, the execution terminates;
- *choice*: a non-deterministic choice within a set of rules. One and only one applicable rule will be executed if such a rule exists, otherwise the whole block is simply skipped;
- *arbitrary*: the rules from the block can be executed zero or more times by non-deterministically choosing any of the applicable rules;
- *maximal parallel*: the classic execution mode used in membrane computing.

These execution strategies and the fact that in any one compartment several blocks with different strategies can be composed and executed offers a lot of flexibility to the kP system designer, similarly to procedural programming.

2.3 kPWorkbench

Kernel P systems are supported by an integrated software suite, kPWorkbench, which employs a set of simulation and formal verification tools and methods that permit simulating and verifying kP system models, written in kP-Lingua.

The verification component of kPWorkbench [5] checks the correctness of kP system models by exhaustively analysing all possible behaviours. In order to facilitate the specification of system requirements, kPWorkbench features a property language, called *kP-Queries*, which comprises a list of property patterns written as natural language statements. The properties expressed in *kP-Queries* are verified using the Spin [13] and NuSMV [3] model checkers after being translated into corresponding *Linear Temporal Logic (LTL)* and *Computation Tree Logic (CTL)* syntax.

The simulation component features a native simulator [2, 16], which allows the users to simulate kP system models efficiently. In addition, kPWorkbench integrates the Flame simulator [4, 19], a general purpose large scale agent based simulation environment, based on a method that allows users to express kP systems as a set of communicating X-machines [11].

2.4 Kernel P systems testing

When testing a kP system model, an automata model needs to be constructed first, based on the computation tree of the kP system. As, in general, the computation tree may be infinite and cannot be modelled by a finite automaton, an approximation of the tree is used. This approximation is obtained by limiting the length of any computation to an upper bound k and considering only computations up to k transitions in length. This approximation is then used to construct a deterministic finite cover automaton (DFCA) of the model [6–8].

However, in the case of the e-bike, this can be naturally modelled by a state-based formalism and, furthermore, the kP system was derived from such a model (Fig. 1). Therefore one can use this state-based model in testing. It can be observed, however, that the model is not exactly a finite automaton since an additional variable is used to decide to which state (PO or PA) the e-bike returns when the Cruise Control facility is cancelled[2]. Such a formalism, that combines a finite state machine like control with data structures is the *stream X-machine* [12].

A stream X-machine (SXM) is like a finite automaton in which the transitions are labelled by partial functions (or, more generally, relations) instead of mere symbols. Formally,

[2] One could build a Finite State Automaton with two extra states (CCPO and CCPA, that allow to come back to PO and PA, respectively, when CC facility is cancelled), plus other necessary transitions from/to these states, in order to simulate the same behaviour of the e-bike model. However, the corresponding X-machine model, having one memory variable instead of the 2 extra states, has the advantage of keeping the control structure simpler; having less states it's easier to be read and the states correspond exactly to the device modes.

Definition 2. *A* stream X-Machine *(abbreviated* SXM*) is a tuple*

$$Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0),$$

where:

- Σ *is the finite* input alphabet*.*
- Γ *is the finite* output alphabet*.*
- Q *is the finite set of* states*.*
- M *is a (possibly infinite) set called* memory*.*
- Φ *is a finite set of distinct* processing functions*; a processing function is a non-empty (partial) function of type* $M \times \Sigma \longrightarrow \Gamma \times M$*.*
- F *is the (partial)* next-state function*,* $F : Q \times \Phi \longrightarrow Q$*.*
- $q_0 \in Q$ *is the initial state.*
- $m_0 \in M$ *is the initial memory value.*

Intuitively, an SXM Z can be thought as a finite automaton with the arcs labelled by functions from the set Φ. The automaton $A_Z = (\Phi, Q, F, q_0)$ over the alphabet Φ is called *the associated finite automaton* (abbreviated *associated FA*) of Z and is usually described by a state-transition diagram. As with any automaton, the function F may be extended to take sequences from Φ^*, to form the function $F^* : Q \times \Phi^* \longrightarrow Q$. We will write $L_{A_Z}(q) = \{p \in \Phi^* \mid (q, p) \in dom\ F^*\}$ to denote the set of paths that can be traced out of state q. When $q = q_0$, this will be called the *language accepted* by Z and denoted L_{A_Z}.

3 kP model for e-bike cruise control

In this section we present a kP system model for the cruise control system described as a state machine in Fig. 1. The corresponding kP system has 2 compartment types: (1) *tEvent*, in charge of generating all possible events (or inputs from the user) and sending them to *tEBike*; (2) *tEBike*, receiving these events and processing them according to state machine rules. The *tEBike* will always contain only one element of the set $\{PO, CC, PC, PA, BR\}$ representing the current state of the machine, and might have other elements such as $\{pa, pc, cc, br, pac, pcc, ccc, brc\}$ representing the event received from *tEvent* or $\{po2cc, pa2cc\}$ as objects recording which was the previous state before CC. The event names are lower case always, compared to their upper case states counterparts, e.g. *pa*, *cc* for pedal assist, cruise control request, while brc, pcc represent brake cancelled or pedal charge cancelled.

Figure 2 presents the KP-Lingua source code corresponding to our model of e-bike cruise control. The execution strategy is *choice* for both compartment types, but in this particular case the maximal parallelism strategy would have provided the same functionality. The computation is infinite and due to the non-determinism of the model we would like to check if some properties hold for any possible computation. The KPL model and verification files discussed here are available for download on the kPWorkbench website[3].

[3] http://kpworkbench.org/index.php/case-studies

```
type tEvent{
 choice{
  g -> g, br(tEBike).
  g -> g, cc(tEBike).
  g -> g, pa(tEBike).
  g -> g, pc(tEBike).
  g -> g, brc(tEBike).
  g -> g, ccc(tEBike).
  g -> g, pac(tEBike).
  g -> g, pcc(tEBike).
  }
}

type tEBike{
 choice{
  PO, br -> BR.
  PO, cc -> CC, po2cc.
  PO, pa -> PA.
  PO, pc -> PC.
  PO, brc -> PO.
  PO, ccc -> PO.
  PO, pac -> PO.
  PO, pcc -> PO.

  PA, br -> BR.
  PA, cc -> CC, pa2cc.
  PA, pa -> PA.
  PA, pc -> PA.
  PA, brc -> PA.
  PA, ccc -> PA.
  PA, pac -> PO.
  PA, pcc -> PA.
  PC, br -> BR.
  PC, cc -> PC.
  PC, pa -> PC.
  PC, pc -> PC.
  PC, brc -> PC.
  PC, ccc -> PC.
  PC, pac -> PC.
  PC, pcc -> PO.

  CC, br, pa2cc -> PA.
  CC, br, po2cc -> PO.
  CC, cc -> CC.
  CC, pa -> CC.
  CC, pc -> CC.
  CC, brc -> CC.
  CC, ccc, po2cc -> PO.
  CC, ccc, pa2cc -> PA.
  CC, pac -> CC.
  CC, pcc -> CC.

  BR, br -> BR.
  BR, cc -> BR.
  BR, pa -> BR.
  BR, pc -> BR.
  BR, brc -> PO.
  BR, ccc -> BR.
  BR, pac -> BR.
  BR, pcc -> BR.
 }
}
cEvent {g} (tEvent).
cEBike {PO} (tEBike).
cEvent - cEBike.
```

Fig. 2. KP-Lingua code for the e-bike cruise control system

4 Verification

In this section, we check various properties of the e-bike model to verify that the model satisfies the system requirements using the verification component of kPWorkbench. The tool translates the kP-Lingua model of the e-bike system into the NuSMV modelling language. Similarly, the properties written in kP-Queries (using natural language statements) are translated into the NuSMV property specification language (the translation can be in LTL or CTL).

Table 1 shows the verification results of the e-bike model properties. The first column shows the property id; the second column describes the properties informally; the third column shows the formal properties expressed in kP-Queries (which are then translated into LTL and CTL in NuSMV syntax); and the last column illustrates the verification result.

The first property checks whether BR is reachable from any state after brake requested. The property holds because BR can be activated directly from PO, PA and PC, and there are paths from CC to BR, too, over PO and PA. The second property verifies that after BR is activated, the system will either stay in BR or move to PO. As expected, this property also holds, because BR cannot request any states other than itself or PO. The properties from 3 to 8 test different transitions from/to the CC state. For example, properties 4 and 5 verify that after CC is cancelled, the system will return to the state from which it was activated, i.e., PO or PA. Properties 3–8 all hold, except for property 8, which is false. This property checks the existence of states (other than PO and PA) from which we may have direct access to the CC state. However, only PO and PA can access CC, so the property does not hold. The remaining properties, 9–12, check the existence/absence of transitions from/to PC. Again, all properties hold except property 11. This property asserts that PC can be activated from a state other than PO, whereas in fact only PO can activate PC. Therefore, it does not hold. The verified properties validate that the e-bike system works as desired.

5 Testing

In this section we show how the kernel P system model from Section 3 can be tested using automata and X-machine based techniques.

For the kP system described in Section 3, the associated stream X-machine (SXM) will be defined as follows:

- the set of states is $Q = \{PO, PA, PC, CC, BR\}$;
- the set of inputs is $\Sigma = \{pa, cc, pc, br, brc, pac, ccc, pcc\}$
- there are no explicit outputs; in order to make the transition observable we consider the output to be the next state for each transition, so the set of outputs is the same as the set of states, $\Gamma = Q$.
- the memory is $M = \{m\}$, $m \in \{\lambda, pa2cc, po2cc\}$ (one memory variable m, where λ represents an undefined value, and $pa2cc, po2cc$ are used to record the last state before enabling the CC feature);
- each processing function is determined by a rewriting rule in *tEBike*, e.g., the $PO, pa \to PA$ rule induces the processing function $\phi_{PO,pa,PA}$ defined by $\phi_{PO,pa,PA}(m, pa) = (PA, m)$, $m \in M$; the $PO, cc \to CC, po2cc$ rule induces processing function $\phi_{PO,cc,CC}(m, cc) = (CC, po2cc)$, $m \in M$; the $CC, ccc, po2cc \to PO$ rule induces processing function $\phi_{CC,ccc,PO}(po2cc, ccc) = (PO, \lambda)$;
- the next-state function is defined by $F(q, \phi_{q,\sigma,q'}) = q'$ for every $q, q' \in Q$, $\sigma \in \Sigma$ such that $\phi_{q,\sigma,q'} \in \Phi$;
- the initial state is $q_0 = PO$;
- the initial memory is $m_0 = \lambda$.

Table 1. Verified properties

#	Description	kP-Queries	Res.
1	Whenever brake is requested, it will eventually be activated.	CTL: cEBike.br >0 **followed-by** cEBike.BR >0	T
2	BR either stays same or it can activate only PO	LTL: **always** ((cEBike.BR >0) **implies** (**next** (cEBike.PO >0 **or** cEBike.BR >0)))	T
3	The user should be able to request / activate CC only from PO or PA	LTL: **never** ((cEBike.BR >0 **or** cEBike.PC >0) **and** (**next** (cEBike.CC >0)))	T
4	If CC activated from PO, then the system will return to PO after CC cancel or brake request	LTL: **always** ((cEBike.CC >0 **and** cEBike.po2cc >0) **and** (cEBike.ccc >0 **or** cEBike.br >0) **implies** (**next**(cEBike.PO >0)))	T
5	If CC activated from PA, then the system will return to PA after CC cancel or brake request	LTL: **always** ((cEBike.CC >0 **and** cEBike.pa2cc >0) **and** (cEBike.ccc >0 **or** cEBike.br >0) **implies** (**next**(cEBike.PA >0)))	T
6	When brake is requested in CC the system returns to PA or PO	LTL: **always** ((cEBike.CC >0 **and** cEBike.br >0) **implies** (cEBike.BR >0 **preceded-by** (cEBike.PO >0 **or** cEBike.PA >0)))	T
7	The system should not transit directly from CC to Brake directly	LTL: **never** ((cEBike.CC >0 **and** cEBike.br >0) **and** (**next** (cEBike.BR >0)))	T
8	CC can be activated from a state other than PO or PA	CTL: (**not** (cEBike.PO >0 **or** cEBike.PA >0)) **until** cEBike.CC >0	F
9	PA and PC cannot directly activate each other	LTL: **never** ((cEBike.PA >0 **and** (**next** (cEBike.PC >0))) **or** (cEBike.PC >0 **and** (**next** (cEBike.PO >0))))	T
10	CC and PC cannot directly activate each other	LTL: **never** ((cEBike.CC >0 **and** (**next** (cEBike.PC >0))) **or** (cEBike.PC >0 **and** (**next** (cEBike.CC >0))))	T
11	PC can be activated from a state other than PO	CTL: (**not** (cEBike.PO >0)) **until** cEBike.PC >0	F
12	PC can activate PC, PO or BR only	LTL: **always** (cEBike.PC >0 **implies** (**next** ((((cEBike.PC >0) **or** (cEBike.PO >0)) **or** (cEBike.BR >0)))))	T

Now, suppose we want to test an implementation of a system specified as an SXM. The testing techniques presented in [12, 14] generate test suites that guarantee that the implementation conforms to the model, provided that some *design for test* conditions are satisfied and the tester is able to estimate the maximum number of states the implementation may have. We denote by β the difference between this estimated upper bound on the number of states of the implementation under test and the number of states of the model.

In order to generate a test suite from a SXM, two set of paths from the associated automaton will have to be constructed: a state cover and a characterisation set.

A *transition cover* of a SXM Z is a set $S \subseteq \Phi^*$ such that for every state $q \in Q$ of Z there is $p \in S$ such that p reaches state q, i.e. $F^*(q_0, p) = q$. In our example, the empty sequence λ reaches the initial state PO, $\phi_{PO,pa,PA}$ reaches PA, $\phi_{PO,pc,PC}$ reaches PC, $\phi_{PO,cc,CC}$ reaches CC and $\phi_{PO,br,BR}$ reaches BR, thus $S = \{\lambda, \phi_{PO,pa,PA}, \phi_{PO,pc,PC}, \phi_{PO,cc,CC}, \phi_{PO,br,BR}\}$ is a state cover of Z.

A *characterization set* of a SXM Z is a set $W \subseteq \Phi^*$ such that for every two distinct states $q, q' \in Q$ there is $p \in W$ such that p distinguishes between q and q', i.e $F^*(q, p)$ is defined and $F^*(q', p)$ is not defined or $F^*(q, p)$ is not defined and $F^*(q', p)$ is defined. In our example, $\phi_{PO,br,BR}$ distinguishes PO from any other state of Z, $\phi_{PA,br,BR}$ distinguishes PA from any other state of Z, $\phi_{PC,br,BR}$ distinguishes PC from any other state of Z and $\phi_{CC,br,PO}$ distinguishes CC from any other state of Z, so $W = \{\phi_{PO,br,BR}, \phi_{PA,br,BR}, \phi_{PC,br,BR}, \phi_{CC,br,PO}\}$ is a characterization set of Z. Once a transition cover and a characterization set have been constructed, the test suite is given by the formula

$$S(\Phi^0 \cup \Phi^1 \cup \ldots \cup \Phi^{\beta+1})W,$$

where S is a transition cover, W is a characterization set, and (as already noted) β denotes the difference between the estimated maximum number of states of the implementation under test and the number of states of the model.

In order for the successful application of the test suite to guarantee the conformance of the implementation to the model, the SXM model has to satisfy two design for test conditions: output-distinguishability and input-completeness. The set of processing functions Φ is called *output-distinguishable* if, for every two processing functions $\phi_1, \phi_2 \in \Phi$, if there exists $m, m_1, m_2 \in M$, $\sigma \in \Sigma$, $\gamma \in \Gamma$ such that $\phi_1(m, \sigma) = (\gamma, m_1)$ and $\phi_2(m, \sigma) = (\gamma, m_2)$ then $\phi_1 = \phi_2$. In our example, Φ is not output-distinguishable since, for example, both $\phi_{PO,br,BR}$ and $\phi_{PA,br,BR}$ produce the output BR while processing any memory value m and input br. The set Φ can be transformed into one that is output-distinguishable by suitably augmenting the output alphabet. In our running example we may enlarge Γ by considering as output for each transition a pair formed by both the current and the next state of the transition.

The set of processing functions Φ is called input-complete if, for every processing function $\phi \in \Phi$ and every memory $m \in M$, there exists an input symbol $\sigma \in \Sigma$ such that (m, σ) is in the domain of ϕ. In our running example, Φ is not input-complete since, for example, for $\phi_{CC,br,PA} \in \Phi$ and $po2cc \in M$, there is no input $\sigma \in \Sigma$ such that $(po2cc, \sigma)$ is in the domain of $\phi_{CC,br,PA}$. The set Φ can be transformed into one that is input-complete by suitably augmenting the input alphabet and the processing functions. In our running example, $\phi_{CC,br,PA}$ can be augmented by introducing an extra input symbol, say σ_e, and setting $\phi_{CC,br,PA}(po2cc, \sigma_e) = (\lambda, PA)$. Naturally, the extra inputs, outputs and transitions will be removed after testing has been completed.

6 Conclusions and Further Work

In this paper, we have presented our current work, focusing on an application of membrane computing to modelling and analysing engineering systems. As our initial attempt, we have considered the cruise control system of electric bike as our case study. We have modelled an e-bike cruise control system using kernel P systems and validated its behaviour using the kPWorkbench verification environment. We have also illustrated how the automata and X-machine testing methodologies can be applied on the kP model of the cruise control system.

As future work, we are planning to show how more complex engineering problems can be solved, tested and verified by using kP systems.

Acknowledgements

The work of SK is supported by Innovate UK (project no: KTP010551). MB is supported by a PhD studentship provided by the Turkey Ministry of Education.

References

1. Bakir, M.E., Ipate, F., Konur, S., Mierla, L., Niculescu, I.: Extended simulation and verification platform for kernel P systems. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) Membrane Computing - 15th International Conference, CMC 2014, Prague, Czech Republic, August 20-22, 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8961, pp. 158–178. Springer (2014), `http://dx.doi.org/10.1007/978-3-319-14370-5_10`
2. Bakir, M.E., Konur, S., Gheorghe, M., Niculescu, I., Ipate, F.: High performance simulations of kernel P systems. In: 2014 IEEE International Conference on High Performance Computing and Communications, 6th IEEE International Symposium on Cyberspace Safety and Security, 11th IEEE International Conference on Embedded Software and Systems, HPCC/CSS/ICESS 2014, Paris, France, August 20-22, 2014. pp. 409–412 (2014), `http://dx.doi.org/10.1109/HPCC.2014.69`
3. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An opensource tool for symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) Computer Aided Verification, 14th International Conference, CAV 2002, Copenhagen, Denmark, July 27-31, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2404, pp. 359–364. Springer (2002), `http://dx.doi.org/10.1007/3-540-45657-0_29`

4. Coakley, S., Gheorghe, M., Holcombe, M., Chin, S., Worth, D., Greenough, C.: Exploitation of high performance computing in the FLAME agent-based simulation framework. In: Min, G., Hu, J., Liu, L.C., Yang, L.T., Seelam, S., Lefèvre, L. (eds.) 14th IEEE International Conference on High Performance Computing and Communication & 9th IEEE International Conference on Embedded Software and Systems, HPCC-ICESS 2012, Liverpool, United Kingdom, June 25-27, 2012. pp. 538–545. IEEE Computer Society (2012), `http://dx.doi.org/10.1109/HPCC.2012.79`
5. Dragomir, C., Ipate, F., Konur, S., Lefticaru, R., Mierla, L.: Model checking kernel P systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing - 14th International Conference, CMC 2013, Chişinău, Republic of Moldova, August 20-23, 2013, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8340, pp. 151–172. Springer (2013), `http://dx.doi.org/10.1007/978-3-642-54239-8_12`
6. Gheorghe, M., Ceterchi, R., Ipate, F., Konur, S.: Kernel P systems modelling, testing and verification - sorting case study. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing - 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10105, pp. 233–250. Springer (2016), `http://dx.doi.org/10.1007/978-3-319-54072-6_15`
7. Gheorghe, M., Ceterchi, R., Ipate, F., Konur, S., Lefticaru, R.: Kernel P systems: from modelling to verification and testing. Theoretical Computer Science (accepted for publication), `http://hdl.handle.net/10454/11720`
8. Gheorghe, M., Ipate, F.: On testing P systems. In: Corne, D.W., Frisco, P., Paun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing - 9th International Workshop, WMC 2008, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers. Lecture Notes in Computer Science, vol. 5391, pp. 204–216. Springer (2008), `http://dx.doi.org/10.1007/978-3-540-95885-7_15`
9. Gheorghe, M., Ipate, F., Dragomir, C., Mierla, L., Valencia-Cabrera, L., García-Quismondo, M., Pérez-Jiménez, M.J.: Kernel P Systems - Version I. Eleventh Brainstorming Week on Membrane Computing (11BWMC) pp. 97–124 (08/2013 2013), `http://www.gcn.us.es/files/11bwmc/097_gheorghe_ipate.pdf`
10. Gheorghe, M., Konur, S., Ipate, F., Mierla, L., Bakir, M.E., Stannett, M.: An integrated model checking toolset for kernel P systems. In: Rozenberg, G., Salomaa, A., Sempere, J.M., Zandron, C. (eds.) Membrane Computing - 16th International Conference, CMC 2015, Valencia, Spain, August 17-21, 2015, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9504, pp. 153–170. Springer (2015), `http://dx.doi.org/10.1007/978-3-319-28475-0_11`
11. Holcombe, M.: X-machines as a basis for dynamic system specification. Software Engineering Journal 3(2), 69–76 (1988), `http://dx.doi.org/10.1049/sej.1988.0009`
12. Holcombe, M., Ipate, F.: Correct Systems: Building a Business Process Solution. Applied computing, Springer-Verlag (1998), `http://dx.doi.org/10.1007/978-1-4471-3435-0`
13. Holzmann, G.J.: The model checker SPIN. IEEE Transactions on Software Engineering 23(5), 275–295 (1997)
14. Ipate, F., Holcombe, M.: An integration testing method that is proved to find all faults. International Journal of Computer Mathematics 63(3-4), 159–178 (1997), `http://dx.doi.org/10.1080/00207169708804559`
15. Konur, S., Gheorghe, M., Dragomir, C., Mierla, L., Ipate, F., Krasnogor, N.: Qualitative and quantitative analysis of systems and synthetic biology constructs using

P systems. ACS Synthetic Biology 4(1), 83–92 (2015), http://dx.doi.org/10.1021/sb500134w, pMID: 25090609

16. Konur, S., Kiran, M., Gheorghe, M., Burkitt, M., Ipate, F.: Agent-based high-performance simulation of biological systems on the GPU. In: 17th IEEE International Conference on High Performance Computing and Communications, HPCC 2015, 7th IEEE International Symposium on Cyberspace Safety and Security, CSS 2015, and 12th IEEE International Conference on Embedded Software and Systems, ICESS 2015, New York, NY, USA, August 24-26, 2015. pp. 84–89. IEEE (2015), http://dx.doi.org/10.1109/HPCC-CSS-ICESS.2015.253
17. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000), http://dx.doi.org/10.1006/jcss.1999.1693
18. The P systems website. http://http://ppage.psystems.eu, [Online; accessed 28/04/17]
19. Sanassy, D., Fellermann, H., Krasnogor, N., Konur, S., Mierla, L., Gheorghe, M., Ladroue, C., Kalvala, S.: Modelling and stochastic simulation of synthetic biological boolean gates. In: 2014 IEEE International Conference on High Performance Computing and Communications, 6th IEEE International Symposium on Cyberspace Safety and Security, 11th IEEE International Conference on Embedded Software and Systems, HPCC/CSS/ICESS 2014, Paris, France, August 20-22, 2014. pp. 404–408 (2014), http://dx.doi.org/10.1109/HPCC.2014.68
20. Varadarajan, A.V., Romijn, M., Oosthoek, B., van de Mortel-Fronczak, J.M., Beijer, J.: Development and validation of functional model of a cruise control system. In: Kofron, J., Tumova, J., Buhnova, B. (eds.) Proceedings of the 13th International Workshop on Formal Engineering Approaches to Software Components and Architectures, FESCA@ETAPS 2016, Eindhoven, The Netherlands, 3rd April 2016. EPTCS, vol. 205, pp. 45–58 (2016), http://dx.doi.org/10.4204/EPTCS.205.4

Solving a Special Case of the P Conjecture Using Dependency Graphs with Dissolution[⋆]

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

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
{leporati, luca.manzoni, mauri, porreca, zandron}@disco.unimib.it

Abstract. We solve affirmatively a new special case of the *P conjecture* by Gh. Păun, which states that P systems with active membranes without charges and without non-elementary membrane division cannot solve **NP**-complete problems in polynomial time. The variant we consider is *monodirectional*, i.e., without send-in communication rules, *shallow*, i.e., with membrane structures consisting of only one level besides the external membrane, and *deterministic*, rather than more generally confluent. We describe a polynomial-time Turing machine simulation of this variant of P systems, exploiting a generalised version of dependency graphs for P systems which, unlike the original version introduced by Cordón-Franco et al., also takes membrane dissolution into account.

1 Introduction

The original variant of P systems with active membranes, which includes membrane charges (or polarisations), solves in polynomial time exactly the problems in the complexity class **PSPACE** [13]. However, the variant without charges appears to be significantly weaker. This led Păun to formulate the P conjecture in 2005 [11, Problem F], one of the long-standing open problems in membrane computing:

> *Can the polarizations be completely avoided?* [...] *The feeling is that this is not possible — and such a result would be rather sound: passing from no polarization (which, in fact, means one polarization) to two polarizations amounts to passing from nonefficiency to efficiency.*

While this general formulation of P conjecture is actually false, as P systems without charges still characterise **PSPACE** when both non-elementary membrane division and dissolution rules are allowed [2], the statement is true when dissolution rules are forbidden [6].

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

The intermediate case, where dissolution is allowed but non-elementary division is not, is still open. The best known upper bound is $\mathbf{P}^{\#\mathbf{P}}$, the class of problems solved in polynomial time by Turing machines with an oracle for a counting problem [7]. However, some restricted special cases actually *do* have a **P** upper bound; this can be proved for P systems having only *symmetric* division rules [9], i.e., of the form $[a]_h \to [b]_h\ [b]_h$, or when the initial membrane structure is linear, and *only* dissolution and elementary division are allowed [14]. We refer the reader to Gazdag and Kolonits [4] for a more detailed survey of related results.

In this paper we consider another special case of the P conjecture, proving a **P** upper bound for P systems with active membranes without charges with the following three restrictions:

- *monodirectional*, that is, without send-in communication rules, as previously investigated for membranes with charges [8];
- *shallow*, that is, having only one level of elementary membranes in addition to the outermost one;
- *deterministic*, that is, having only one computation, instead of having multiple computations with the same result as in the usual *confluent* mode.

We believe that these constraints are quite natural and interesting, since monodirectional shallow deterministic P systems with active membranes have long been known to solve **NP**-complete problems in polynomial time with only two membrane charges [1]. Furthermore, they have been recently [8] proved to characterise $\mathbf{P}_{\|}^{\mathbf{NP}}$, which allows parallel queries to an **NP** oracle, with three charges[1]; clearly, the $\mathbf{P}_{\|}^{\mathbf{NP}}$ upper bound also applies to the variant without charges considered in this paper.

Among the three assumptions above, the most fundamental one is monodirectionality. As we will prove later, this restriction implies that the result of the computation only depends on at most one elementary membrane, although establishing *which one* is nontrivial. We hope, instead, to relax the latter two constraints in future works.

Besides the **P** upper bound itself, the main contribution of this paper is the tool we exploit in order to prove the upper bound. This is a generalisation of the *dependency graphs*, introduced by Cordón-Franco et al. [3] to establish the result of P systems without charges and without dissolution rules, to P systems which do include dissolving membranes. This generalisation has the potential to be extended to other variants of P systems for proving upper bounds to the complexity classes they characterise.

2 Basic Notions

The P systems analysed in this paper are *monodirectional* P systems with active membranes [8] without charges [6], using object evolution rules $[a \to w]_h$, send-

[1] The determinism of the P systems is not explicitly stated in the original paper [8], but can be easily checked by inspection of the rules.

out communication rules $[a]_h \to [\ ]_h\ b$, membrane dissolution rules $[a]_h \to b$ and elementary membrane division rules $[a]_h \to [b]_h\ [c]_h$. These P systems do not use send-in communication rules $a\ [\ ]_h \to [b]_h$, or division rules for non-elementary membranes, either of the "weak" form $[a]_h \to [b]_h\ [c]_h$ or the "strong" form $\big[[\ ]_k\ [\ ]_\ell\big]_h \to \big[[\ ]_k\big]_h\ \big[[\ ]_\ell\big]_h$.

Furthermore, we focus on *shallow* P systems, which have membrane structures of depth at most 1, that is, at most one level of elementary membranes besides the outermost membrane.

Finally, we require our P systems to be *deterministic*, that is, each configuration reachable from the initial one has at most one successor configuration. Notice that this condition, although much stronger than the usual confluence requirement (where multiple computations can exist, as long as they all agree on the result), does not require a single multiset of rules to be applicable at each computation step, but only that all applicable multisets of rules produce the same result.

For brevity, in this paper we refer to monodirectional, shallow, deterministic P systems as MSD P systems.

In particular, we are dealing with *recogniser P systems* [12], whose alphabet includes the distinguished result objects yes and no; exactly one result object must be sent out from the outermost membrane to signal acceptance or rejection, and only at the last computation step.

A decision problem, or language $L \subseteq \Sigma^\star$, is solved by a *family* of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$, where Π_x accepts if and only if $x \in L$. In that case, we write $L(\boldsymbol{\Pi}) = L$. As usual, we require a uniformity condition [10] on families of P systems:

Definition 1. *A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is* (polynomial-time) uniform *if the mapping $x \mapsto \Pi_x$ can be computed by two polynomial-time deterministic Turing machines E and F as follows:*

- *$F(1^n) = \Pi(n)$ is a common P system for all inputs of length n, with a distinguished input membrane.*
- *$E(x) = w_x$ is an input multiset for $\Pi(|x|)$, encoding the specific input x.*
- *Finally, Π_x is simply $\Pi(|x|)$ with w_x added to its input membrane.*

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

We denote the class of problems solved by uniform (resp., semi-uniform) families of deterministic P systems of type $\mathcal{D}$ as $\mathbf{DMC}_\mathcal{D}$ (resp., $\mathbf{DMC}^\star_\mathcal{D}$). We denote the corresponding class of problems solved *in polynomial time* by $\mathbf{DPMC}_\mathcal{D}$ (resp., $\mathbf{DPMC}^\star_\mathcal{D}$).

3 Properties of MSD P Systems

We begin our analysis of MSD P systems by proving that their overall behaviour, acceptance or rejection, is actually governed by *just one or two objects*, which must moreover be located inside a single membrane. In order to be able to succinctly formalise this result, we first define a notion of restricted configuration similar, but more general than the one previously used by the authors [8, Definition 3].

Definition 2. *Given two configurations $\mathcal{C}$, $\mathcal{D}$ of a P system, we say that $\mathcal{C}$ is a* restriction *of $\mathcal{D}$, in symbols $\mathcal{C} \sqsubseteq \mathcal{D}$, if $\mathcal{C}$ is obtained from $\mathcal{D}$ by deleting zero or more membranes, including their whole content (both objects and children membranes), and zero or more of the remaining objects.*

Being based on the subtree partial ordering of membrane structures and on the submultiset partial order, the relation $\sqsubseteq$ is also a partial order.

For the purposes of this paper, we consider as valid restricted configurations even those obtained by only keeping part of the environment and ignoring the membrane structure altogether. For instance, given the configuration

$$\mathcal{D} = \big[[a\ b]_k\ [c\ c]_\ell\ d\ d\ d\big]_h\ e\ f$$

the following are all valid restrictions of $\mathcal{D}$:

$$\mathcal{C}_1 = [[a\ b]_k\ d\ d]_h \qquad \mathcal{C}_2 = [[c\ c]_\ell]_h \qquad \mathcal{C}_3 = [d\ d\ d]_h\ e \qquad \mathcal{C}_4 = e\ f$$

A subclass of restricted configurations that we will focus on in this paper consists of the "small" configurations, which contain only one object, or up to two objects, if they are both located inside an internal membrane.

Definition 3. *We call a configuration $\mathcal{C}$ of an MSD P system Π a* small configuration *if it consists of the isolated object* yes *or* no *in the environment, or if it is of one of the forms $[a]_h$, $\big[[a]_k\big]_h$, or $\big[[a\ b]_k\big]_h$ where h is the outermost membrane of Π, k is the label of an internal membrane, and a, b are objects of the alphabet.*

A simple counting argument shows that the number of small configurations for an MSD P system is bounded by $(m^2 + m)\ell + 2 \in O(m^2\ell)$, where m is the size of the alphabet, and ℓ the number of labels, which corresponds to the initial number of membranes.

The following result shows that a halting computation of an MSD P system contains a sequence of small configurations which, alone, suffice to establish the result of the computation. This theorem is a stronger version of an analogous result for monodirectional P systems with charges [8, Lemma 1], where a *polynomial* number of objects and membranes were necessary and sufficient to decide the result of the computation.

Theorem 1. *Let Π be an accepting (resp., rejecting) MSD P system, and let $\mathcal{C}_1$ be a configuration reachable in any number of steps from the initial configuration $\mathcal{C}_0$ of Π. Let $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_t)$ be the halting subcomputation starting at $\mathcal{C}_1$. Then, there exists a sequence of small configurations $\boldsymbol{\mathcal{D}} = (\mathcal{D}_1, \ldots, \mathcal{D}_t)$ with $\mathcal{D}_i \sqsubseteq \mathcal{C}_i$ for $1 \leq i \leq t$ and $\mathcal{D}_t =$ yes (resp., $\mathcal{D}_t =$ no) and a sequence of configurations $\boldsymbol{\mathcal{E}} = (\mathcal{E}_2, \ldots, \mathcal{E}_t)$ such that $\mathcal{D}_i \to \mathcal{E}_{i+1}$ and $\mathcal{D}_{i+1} \sqsubseteq \mathcal{E}_{i+1} \sqsubseteq \mathcal{C}_{i+1}$ for all $1 \leq i < t$.*

Proof. We construct the sequences $\boldsymbol{\mathcal{D}} = (\mathcal{D}_1, \dots, \mathcal{D}_t)$ and $\boldsymbol{\mathcal{E}} = (\mathcal{E}_2, \dots, \mathcal{E}_t)$ by recursion on t. If $t = 1$ then $\mathcal{C}_1$ is already an accepting (resp., rejecting) configuration, and thus $\mathcal{D}_1 = \mathsf{yes} \sqsubseteq \mathcal{C}_1$ (resp, $\mathcal{D}_1 = \mathsf{no} \sqsubseteq \mathcal{C}_1$); in this case, the sequence $\boldsymbol{\mathcal{E}}$ is empty.

Now suppose $t > 1$. Then, the sub-computation $\boldsymbol{\mathcal{C}}' = (\mathcal{C}_2, \dots, \mathcal{C}_t)$, i.e., the same computation as $\boldsymbol{\mathcal{C}}$ but starting from the second step, is a halting computation starting at a configuration reachable from the initial configuration $\mathcal{C}_0$ of Π. By induction hypothesis, there exists a sequence $\boldsymbol{\mathcal{D}}' = (\mathcal{D}_2, \dots, \mathcal{D}_t)$ of small configurations and a sequence $\boldsymbol{\mathcal{E}}' = (\mathcal{E}_3, \dots, \mathcal{E}_t)$ of configurations satisfying the statement of the theorem.

We construct $\mathcal{D}_1$ and $\mathcal{E}_2$ according to the form of $\mathcal{D}_2$ and the choice of rules that may have produced that configuration. The following list exhausts all possibilities.

(1) If $\mathcal{D}_2 = \mathsf{yes}$ (resp., $\mathcal{D}_2 = \mathsf{no}$), then there necessarily exists a send-out rule $[a]_h \to [\;]_h\ \mathsf{yes}$ (resp., $[a]_h \to [\;]_h\ \mathsf{no}$) which is applied in the step $\mathcal{C}_1 \to \mathcal{C}_2$. In this case, we let $\mathcal{D}_1 = [a]_h$ and $\mathcal{E}_2 = [\;]_h\ \mathsf{yes}$ (resp., $\mathcal{E}_2 = [\;]_h\ \mathsf{no}$).

(2) If $\mathcal{D}_2 = [a]_h$ and object a is produced by an object evolution rule $[b \to a\ w]_h$, then let $\mathcal{D}_1 = [b]_h$ and $\mathcal{E}_2 = [a\ w]_h$.

(3) If $\mathcal{D}_2 = [a]_h$ and object a is produced by a send-out rule $[b]_k \to [\;]_k\ a$, then let $\mathcal{D}_1 = \big[[b]_k\big]_h$ and $\mathcal{E}_2 = [[\;]_k\ a]_h$.

(4) If $\mathcal{D}_2 = [a]_h$ and object a is produced by a dissolution rule $[b]_k \to a$, then let $\mathcal{D}_1 = \big[[b]_k\big]_h$ and $\mathcal{E}_2 = [a]_h$.

(5) If $\mathcal{D}_2 = [a]_h$ and object a appeared inside an internal membrane k in $\mathcal{C}_1$, but fell out due to another object b applying a dissolution rule $[b]_k \to c$, then let $\mathcal{D}_1 = \big[[a\ b]_k\big]_h$ and $\mathcal{E}_2 = [a\ c]_h$.

(6) If $\mathcal{D}_2 = [a]_h$ and object a evolved from an object b appearing inside an internal membrane k in $\mathcal{C}_1$ using the rule $[b \to a\ w]_k$, and fell out due to another object c applying a dissolution rule $[c]_k \to d$, then let $\mathcal{D}_1 = \big[[b\ c]_k\big]_h$ and $\mathcal{E}_2 = [a\ w\ d]_h$.

(7) If $\mathcal{D}_2 = \big[[a]_k\big]_h$ and object a is produced by an evolution rule $[b \to a\ w]_k$, then let $\mathcal{D}_1 = \big[[b]_k\big]_h$ and $\mathcal{E}_2 = \big[[a\ w]_k\big]_h$.

(8) If $\mathcal{D}_2 = \big[[a]_k\big]_h$ and object a is produced by a division rule $[c]_k \to [a]_k\ [b]_k$, then let $\mathcal{D}_1 = \big[[c]_k\big]_h$ and $\mathcal{E}_2 = [[a]_k\ [b]_k]_h$.

(9) If $\mathcal{D}_2 = \big[[a\ b]_k\big]_h$ and objects a, b are produced by an object evolution rule $[c \to a\ b\ w]_k$, then let $\mathcal{D}_1 = \big[[c]_k\big]_h$ and $\mathcal{E}_2 = \big[[a\ b\ w]_k\big]_h$.

(10) If $\mathcal{D}_2 = \big[[a\ b]_k\big]_h$ and objects a, b are produced by two object evolution rules $[c \to a\ v]_k$ and $[d \to b\ w]_k$, then let $\mathcal{D}_1 = \big[[c\ d]_k\big]_h$ and $\mathcal{E}_2 = \big[[a\ v\ b\ w]_k\big]_h$.

(11) If $\mathcal{D}_2 = \big[[a\ b]_k\big]_h$, object a is produced by an object evolution rule $[c \to a\ w]_k$, and object b already appeared inside membrane k, then let $\mathcal{D}_1 = \big[[c\ b]_k\big]_h$ and $\mathcal{E}_2 = \big[[a\ w\ b]_k\big]_h$.

(12) If $\mathcal{D}_2 = \big[[a\ b]_k\big]_h$, object a is produced by a division rule $[c]_k \to [a]_k\ [d]_k$, and object b is produced by an object evolution rule $[e \to b\ w]_k$, then let $\mathcal{D}_1 = \big[[c\ e]_k\big]_h$ and $\mathcal{E}_2 = [[a\ b\ w]_k\ [d\ b\ w]_k]_h$.

$$\begin{tikzcd}
\mathcal{C}_0 \arrow[r] \arrow[d, squiggly] & \mathcal{C}_1 \arrow[r] \arrow[d, phantom, "\sqsupseteq" sloped] & \mathcal{C}_2 \arrow[rr, dashed] \arrow[d, phantom, "\sqsupseteq" sloped] & & \mathcal{C}_{t-1} \arrow[r] \arrow[d, phantom, "\sqsupseteq" sloped] & \mathcal{C}_t \arrow[d, phantom, "\sqsupseteq" sloped] \\
\mathcal{D}_0 \arrow[dr] & \mathcal{D}_1 \arrow[dr] & \mathcal{D}_2 \arrow[dr, dash, dashed] & \arrow[d, dash, dashed] \arrow[dr, dashed] & \mathcal{D}_{t-1} \arrow[dr] & \mathcal{D}_t \\
 & \mathcal{E}_1 \arrow[u, squiggly] & \mathcal{E}_2 \arrow[u, squiggly] & {} & \mathcal{E}_{t-1} \arrow[u, squiggly] & \mathcal{E}_t \arrow[u, squiggly]
\end{tikzcd}$$

Fig. 1. A computation $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{t-1}, \mathcal{C}_t)$ of Π and a sequence of configurations $\mathcal{C}_0 \rightsquigarrow \mathcal{D}_0 \to \mathcal{E}_1 \rightsquigarrow \mathcal{D}_1 \to \mathcal{E}_2 \rightsquigarrow \mathcal{D}_2 \to \cdots \to \mathcal{E}_{t-1} \rightsquigarrow \mathcal{D}_{t-1} \to \mathcal{E}_t \rightsquigarrow \mathcal{D}_t$ alternating restriction steps ($\rightsquigarrow$) and application of the rules from the corresponding steps of $\boldsymbol{\mathcal{C}}$ ($\to$) limited to the objects in $\mathcal{D}_i$. The diagram "commutes" in the sense that, starting from $\mathcal{C}_0$, the two paths either both lead to accepting configurations, or both to rejecting configurations. Notice that $\mathcal{E}_i \sqsubseteq \mathcal{C}_i$ for all i, since $\mathcal{E}_i$ is obtained from $\mathcal{D}_{i-1} \sqsubseteq \mathcal{C}_{i-1}$ and the P system is deterministic.

(13) If $\mathcal{D}_2 = \big[[a\ b]_k\big]_h$, object a is produced by a division rule $[c]_k \to [a]_k\ [d]_k$, and object b already appeared inside membrane k, then let $\mathcal{D}_1 = \big[[c\ b]_k\big]_h$ and $\mathcal{E}_2 = [[a\ b]_k\ [d\ b]_k]_h$.

It is easy to check by inspection that in all cases (1)–(13) we have $\mathcal{D}_1 \sqsubseteq \mathcal{C}_1$, $\mathcal{D}_1 \to \mathcal{E}_2$, and $\mathcal{D}_2 \sqsubseteq \mathcal{E}_2 \sqsubseteq \mathcal{C}_2$ as required. □

Fig. 1 shows the relationship between the computation $\boldsymbol{\mathcal{C}}$ and the sequences of configurations $\boldsymbol{\mathcal{D}}$ and $\boldsymbol{\mathcal{E}}$ in the statement of Theorem 1.

Notice that Theorem 1 does not directly provide an algorithm for computing *which* sequence $\boldsymbol{\mathcal{D}} = (\mathcal{D}_0, \ldots, \mathcal{D}_t)$ of small configurations gives the result of the computation. Furthermore, as will be shown in Section 4, it is not even sufficient to check that $\mathcal{D}_0 \sqsubseteq \mathcal{C}_0$ and $\mathcal{D}_t = \mathsf{yes}$ to guarantee the acceptance of Π, due to the possible interference from objects in $\boldsymbol{\mathcal{C}}$ not appearing in $\boldsymbol{\mathcal{D}}$.

Another fundamental limitation of MSD P systems is that they always halt in linear time with respect to the size of the alphabet, otherwise they would enter an infinite loop.

Theorem 2. *Every MSD P system halts within* $2m$ *steps, where* m *is the size of its alphabet.*

Proof. Assume that the P system is accepting, as the rejecting case is symmetrical. The object yes descends from an object a contained in the initial configuration of the P system via a sequence of rule applications. The object a is initially either inside the outermost membrane, or inside an elementary membrane immediately contained therein; in the latter case, either it reaches the outermost membrane via communication or by dissolving the membrane, or it reaches the outermost membrane due to *another* object dissolving the membrane.

If a is initially located inside the outermost membrane, then it can be rewritten a number of times by object evolution rules until an object b with an associated send-out rule $[b]_h \to [\,]_h$ yes is produced; notice that object evolution is the only way to produce b, as the outermost membrane cannot divide. In the shortest possible computation, the object b appears after a sequence of at most $m-1$ rewriting steps, where at least one new object is introduced at each step: indeed, since object evolution rules are context-free, each object either (i) becomes b within $m-1$ steps, or (ii) it stops evolving before $m-1$ steps without ever becoming b, or (iii) it enters an infinite rewriting loop. Case (iii) is impossible, since the P system is a recogniser; case (ii) is also impossible, since reaching b is necessary in order to send out the result yes. Hence case (i) must hold, and the computation accepts in at most m steps.

An analogous argument shows that a rule of the form $[a]_k \to [\,]_k\ b$ or $[a]_k \to b$ is applied inside the elementary membrane containing a after at most $m-1$ steps, if a is not initially located inside the outermost membrane. After further $m-1$ steps, the result is then sent out by a rule $[c]_h \to [\,]_h$ yes as detailed above. In this case, the accepting computation has a length at most $2m$. □

This result implies that the class of languages recognised by MSD P systems with no restriction on computation resources is the same as the class of languages they recognise in polynomial time.

Corollary 1. $\mathbf{DPMC}_{\mathcal{D}} = \mathbf{DMC}_{\mathcal{D}}$ *and* $\mathbf{DPMC}^{\star}_{\mathcal{D}} = \mathbf{DMC}^{\star}_{\mathcal{D}}$*, where* $\mathcal{D}$ *is the class of MSD P systems.* □

4 Dependency Graphs with Dissolution

Dependency graphs for P systems were introduced by Cordón-Franco et al. [3] as a way to track the objects in a P system without charges with non-cooperative rules and fixed membrane structures, and were later extended to membrane division rules [5], leaving out only membrane dissolution.

A dependency graph for a P system Π of this kind has, as vertices, all possible configurations of the form $\big[[\cdots\ [a]_{h_d}\ \cdots]_{h_1}\big]_{h_0}$, consisting of a linear sub-membrane structure of the membrane structure of Π, and a single object contained in the innermost membrane; a single object a in the environment (i.e., without any surrounding membrane) is also allowed[2]. Two configurations $\mathcal{V}, \mathcal{W}$ of the dependency graph are connected by an oriented edge if, when applying the rules of Π from configuration $\mathcal{V}$ (recall that, for a confluent P system, multiple choices are generally possible), we can reach a configuration $\mathcal{V}'$ containing $\mathcal{W}$, or $\mathcal{W} \sqsubseteq \mathcal{V}'$ in our notation (Definition 2).

[2] In the original notation [3,5] the surrounding membranes are left implicit, and thus the vertex $\big[[\cdots\ [a]_{h_d}\ \cdots]_{h_1}\big]_{h_0}$ is simply denoted by (a, h_d); an object a in the environment is denoted by (a, env).

The usefulness of the dependency graph for a P system Π without dissolution lies in the fact that it can be constructed in polynomial time from the description of Π, and that Π accepts if and only if there exists a vertex $\mathcal{V}$ contained in the initial configuration $\mathcal{C}_0$, in symbols $\mathcal{V} \sqsubseteq \mathcal{C}_0$, that is connected with a path to the node yes, representing the positive answer object in the environment[3]. This property can be easily checked in polynomial time, and this proves that the P conjecture is indeed true *limited to P systems without dissolution* [5].

The reason why checking reachability on dependency graphs suffices is due to the fact that such P systems lack *cooperation*, or, in other words, the only interaction between objects is due to the fact that the rules (excluding object evolution) can compete for the same membrane. Indeed, a theorem analogous to our Theorem 1, but stating that the result of the entire computation depends only on "very small configurations" containing *exactly one single object*, can be easily inferred from the above-mentioned result [5].

Allowing membrane dissolution breaks the entire reasoning, by introducing cooperation (or context-sensitiveness). Indeed, consider a configuration $\big[[a\ u]_k\big]_h$ with a dissolution rule $[a]_k \to b$ and zero or more object evolution rules rewriting the multiset u into v. This configuration leads to $[b\ v]_h$ in one step, and now the objects in v are subject to a potentially completely different set of rules, since the label of the membrane containing them was changed by the object a dissolving k. Thus, the objects in v have been affected by a, which is not possible in a P system without charges and without dissolution; furthermore, the influence of a on v cannot be represented by a standard dependency graph, as this tracks each object separately.

A way to generalise dependency graphs in order to take cooperation into account is to use larger configurations as vertices, containing multiple objects; as an extreme case, the whole set of configurations of Π could be kept as set of vertices. The side effect of this choice is the growth of the dependency graph. In general, a dependency graph keeping track of n objects per vertex has $\Theta(n^m)$ vertices, where m is the size of the alphabet; this value is exponential if $n \in \Theta(m)$. If there is no bound on the number of objects per vertex, the dependency graph might even become infinite.

In the case of MSD P systems, however, we can exploit Theorem 1 to keep the vertices of the dependency graph small, and thus reducing their number to polynomial, since at most two objects per vertex are needed even in the presence of dissolution.

Definition 4. *The* dependency graph $G(\Pi) = \big(V(\Pi), E(\Pi)\big)$ *for an MSD P system has as vertices* $V(\Pi)$ *all small configurations of* Π *and, as edges, the set* $E(\Pi) = \big\{(\mathcal{U}, \mathcal{V}) : \mathcal{U} \to \mathcal{C}$ *for some configuration* $\mathcal{C}$ *and* $\mathcal{V} \sqsubseteq \mathcal{C}\big\}$. *We denote by* $Y(\Pi)$ *the subset of* $V(\Pi)$ *consisting of all vertices connected with a path to the small configuration* yes.

Fig. 2 shows the dependency graph for a simple MSD P system using all available types of rule.

[3] That is, node $(\mathsf{yes}, \mathrm{env})$ in the original notation.

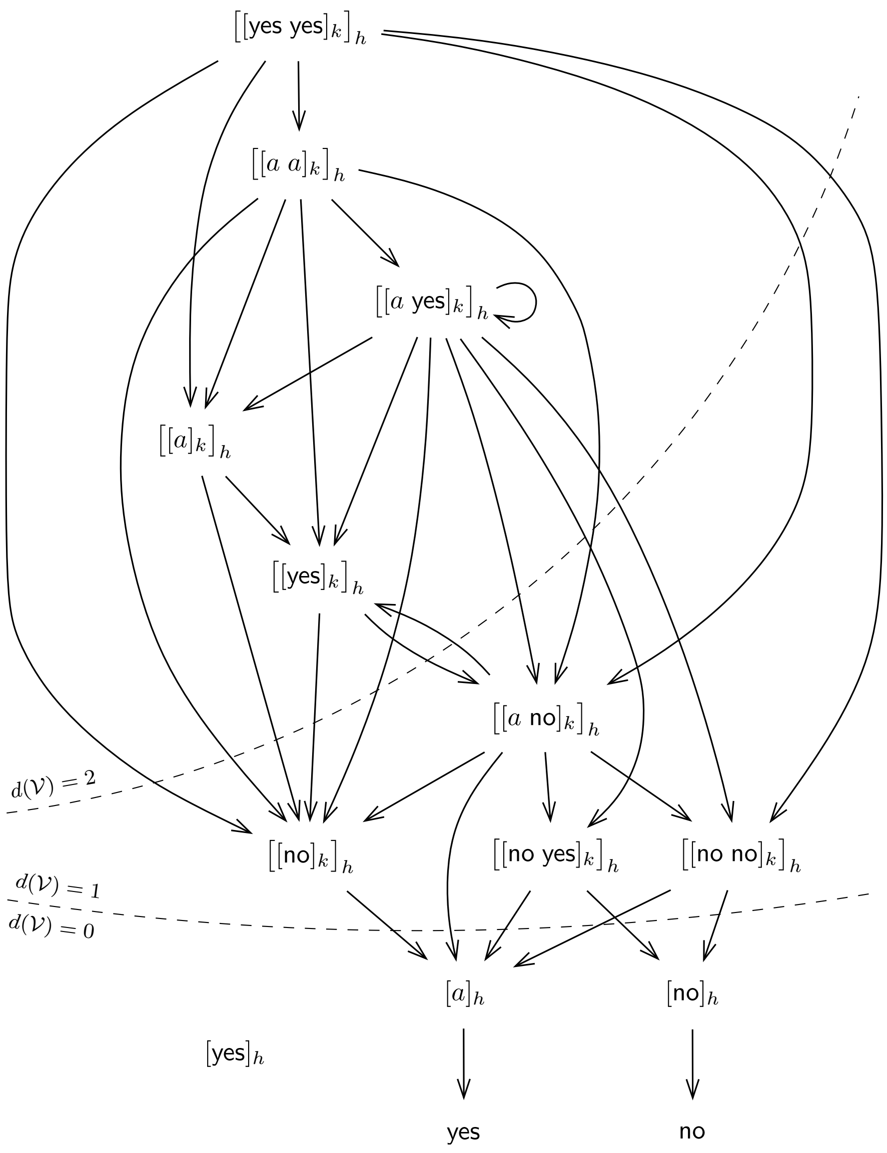


Fig. 2. Dependency graph for an MSD P system Π having alphabet $\Gamma = \{a, \text{yes}, \text{no}\}$ and rules $[a]_k \to [\text{yes}]_k\ [\text{no}]_k$, $[\text{no}]_k \to a$, $[\text{yes} \to a\ \text{no}]_k$, $[\text{no}]_h \to [\ ]_h$ no, $[a]_h \to [\ ]_h$ yes. Notice that this dependency graph is neither connected (the vertex $[\text{yes}]_h$ is isolated), nor acyclic (it even includes a self-loop). Also notice that some vertices, such as $[[a\ \text{no}]_k]_h$, show nondeterministic behaviour, since two rules can be applied; this does not necessarily contradict the determinism of Π, since these vertices $\mathcal{V}$ might not satisfy $\mathcal{V} \sqsubseteq \mathcal{C}$ for any configuration $\mathcal{C}$ reachable from the initial configuration. This P system accepts, for instance, from the initial configuration $\mathcal{C}_0 = [[\text{no}]_k]_h$, but rejects from $\mathcal{C}_0 = [\text{no}]_h$. The dashed lines are isolines grouping together the vertices by distance d (Definition 5).

Notice that $\mathsf{yes} \sqsubseteq \mathcal{C}$ if and only if the object yes is located in the environment of configuration $\mathcal{C}$, that is, if and only if $\mathcal{C}$ is an accepting configuration. Hence, the small configurations $\mathcal{V} \in Y(\mathit{\Pi})$ are those that allow a configuration $\mathcal{C} \sqsupseteq \mathcal{V}$ to ultimately reach an accepting configuration, under certain conditions. Unfortunately, the mere inclusion $\mathcal{V} \sqsubseteq \mathcal{C}$, which allowed us to establish the result of P systems without dissolution, does not suffice in the case of MSD P systems, as shown by the following example.

Example 1. Consider an MSD P system $\mathit{\Pi}$ with the rules

$$\begin{array}{cccc} [a \to c]_k & [b \to d]_k & [c \to e]_k & [d]_k \to f \\ [c]_h \to [\,]_h \ \mathsf{no} & [e]_h \to [\,]_h \ \mathsf{yes} & [g]_h \to f & \end{array}$$

The dependency graph for $\mathit{\Pi}$ includes the following path:

$$\boldsymbol{\mathcal{D}} = \big(\big[[a\ b]_k\big]_h, \big[[c\ d]_k\big]_h, [e]_h, \mathsf{yes}\big)$$

If the initial configuration of $\mathit{\Pi}$ is $\mathcal{C}_0 = \big[[a\ b]_k\big]_h$, then $\mathit{\Pi}$ has indeed the accepting computation

$$\big[[a\ b]_k\big]_h \quad \to \quad \big[[c\ d]_k\big]_h \quad \to \quad [e\ f]_h \quad \to \quad [f]_h \ \mathsf{yes}$$

If, on the other hand, the initial configuration is $\mathcal{C}_0 = \big[[a\ b\ g]_k\big]_h$, i.e., also including the object g, the resulting computation is rejecting:

$$\big[[a\ b\ g]_k\big]_h \quad \to \quad [c\ d\ f]_h \quad \to \quad [d\ f]_h \ \mathsf{no}$$

This shows that it is not sufficient to find a vertex $\mathcal{V}$, in this case $\mathcal{V} = \big[[a\ b]_k\big]_h$, such that $\mathcal{V} \sqsubseteq \mathcal{C}_0$ and there exists a path from $\mathcal{V}$ to yes if dissolution rules are allowed, since objects such as d may interfere with the path leading to yes.

As a consequence, the use of small configurations alone does not suffice to correctly determine the result of P systems where dissolution is allowed. A property more complex than "being connected via a path to the vertex yes" needs to be checked for the vertices contained in the initial configuration $\mathcal{C}_0$. The rest of this section is devoted to establishing this property.

We refer to objects such as g in Example 1, which do *not* appear in a small configuration $\mathcal{V}$ connected by a path to the vertex yes, and yet interfere with that path, as *troublemakers for* $\mathcal{V}$. In order to give a formal definition of troublemakers, we first define a few related notions.

First of all, notice that when the current configuration of the P system contains a vertex $\mathcal{V}$ of the form $[a]_h$, with an object located inside the outermost membrane, then no troublemaker can exist, since the outermost membrane cannot be dissolved. This applies, of course, also when $\mathcal{V} = \mathsf{yes}$, that is, after having reached the halting configuration. For the remaining vertices, we define a notion of distance from such vertices without troublemakers.

Definition 5. *Let Π be an MSD P system; define the function $d\colon Y(\Pi) \to \mathbb{N}$ by letting $d(\mathcal{V}) = 0$ if $\mathcal{V} = \mathsf{yes}$ or $\mathcal{V} = [a]_h$ for some $a \in \Gamma$, and letting $d(\mathcal{V})$ be the distance (in terms of oriented edges) of $\mathcal{V}$ to the nearest small configuration of the form $[a]_h$ in the subgraph of $G(\Pi)$ induced by $Y(\Pi)$, otherwise.*

Fig. 2 also shows the distances of each node of the example dependency graph.

Among the troublemakers for a small configuration of the form $\mathcal{V} = \big[[a]_k\big]_h$ or $\mathcal{V} = \big[[a\ b]_k\big]_h$ there are the objects that dissolve membrane k, unless a or b themselves have dissolution rules. We refer to these objects with the symbol

$$\mathrm{dis}(k) = \{a \in \Gamma : \text{there exists a rule } [a]_k \to b \text{ for some } b \in \Gamma\}$$

Other troublemakers do not dissolve membrane k immediately, but rather produce the set of objects X that are troublemakers for vertices reachable from $\mathcal{V}$ with an edge. These may be produced either by object evolution rules, and we denote them by

$$\mathrm{evo}_k(X) = \{a \in \Gamma : \text{there exists a rule } [a \to b\ w]_k \text{ for some } b \in X\}$$

or by division rules, and we denote them by

$$\mathrm{div}_k(X) = \{a \in \Gamma : \text{there exists a rule } [a]_k \to [b]_k\ [c]_k \text{ for some } b, c \in X\}$$

Notice that send-out rules cannot produce troublemakers for any connected vertex, since the outermost membrane, where the result of the send-out appears, cannot be dissolved.

Now let $\mathcal{V}$ be a small configuration of the form $\big[[a]_k\big]_h$ or $\big[[a\ b]_k\big]_h$. In order to recursively compute its set of troublemakers, let us consider all vertices $\mathcal{W}$ with an edge $(\mathcal{V}, \mathcal{W}) \in E(\Pi)$. Among these, we keep only the vertices $\mathcal{W}_1, \ldots, \mathcal{W}_n$ having distance $d(\mathcal{W}_i) = d(\mathcal{V}) - 1$. The reason to exclude adjacent vertices $\mathcal{W}$ with $d(\mathcal{W}) \geq d(\mathcal{V})$ is that these require more computation steps to reach a troublemaker-free small configuration and, intuitively, the set of troublemakers of a vertex can increase with its distance. Furthermore, the troublemakers of $\mathcal{V}$ only depend on the *intersection* of the troublemakers of $\mathcal{W}_1, \ldots, \mathcal{W}_n$; indeed, a troublemaker for $\mathcal{W}_i$ that is not simultaneously a troublemaker for some $\mathcal{W}_j$ can be bypassed by following the edge $(\mathcal{V}, \mathcal{W}_j)$.

Based on this intuition we can now define, for each small configuration $\mathcal{V}$ on a path to yes, the notion of *set of troublemakers* $\mathrm{tm}(\mathcal{V})$ by recursion on the distance $d(\mathcal{V})$.

Definition 6. *Let Π be an MSD P system and let 2^Γ be the power set of its alphabet. Define the function $\mathrm{tm}\colon Y(\Pi) \to 2^\Gamma$ as*

$$\mathrm{tm}(\mathcal{V}) = \varnothing \tag{1}$$

if $d(\mathcal{V}) = 0$ or $d(\mathcal{V}) = 1$, and

$$\mathrm{tm}(\mathcal{V}) = \mathrm{dis}(k) \cup \mathrm{evo}_k\Big(\bigcap_{i=1}^{n} \mathrm{tm}(\mathcal{W}_i)\Big) \cup \mathrm{div}_k\Big(\bigcap_{i=1}^{n} \mathrm{tm}(\mathcal{W}_i)\Big) \tag{2}$$

if $d(\mathcal{V}) \geq 2$*, where* $\mathcal{W}_1, \ldots, \mathcal{W}_n$ *are all vertices in* $Y(\Pi)$ *such that* $(\mathcal{V}, \mathcal{W}_i) \in E(\Pi)$ *and* $d(\mathcal{W}_i) = d(\mathcal{V}) - 1$.

In fact, we can actually prove that the set of troublemakers $\mathrm{tm}(\mathcal{V})$ depends only on the distance $d(\mathcal{V})$ and on the label of the internal membrane, but not on the actual objects it contains.

Lemma 1. *Let* Π *be an MSD P system, let* $\mathcal{U}, \mathcal{V}$ *be two small configurations of* Π *such that, if both* $\mathcal{U}$ *and* $\mathcal{V}$ *contain two nested membranes, then the internal ones have the same label, and assume that* $d(\mathcal{U}) = d(\mathcal{V})$*. Then* $\mathrm{tm}(\mathcal{U}) = \mathrm{tm}(\mathcal{V})$.

Proof. By induction on $d(\mathcal{U})$. If $d(\mathcal{U}) = d(\mathcal{V}) = 0$ or $d(\mathcal{U}) = d(\mathcal{V}) = 1$, then $\mathrm{tm}(\mathcal{U}) = \varnothing = \mathrm{tm}(\mathcal{V})$. If $d(\mathcal{U}) = d(\mathcal{V}) \geq 2$ then, according to equation (2), we have

$$\mathrm{tm}(\mathcal{U}) = \mathrm{dis}(k) \cup \mathrm{evo}_k\left(\bigcap_{i=1}^{n} \mathrm{tm}(\mathcal{W}_i)\right) \cup \mathrm{div}_k\left(\bigcap_{i=1}^{n} \mathrm{tm}(\mathcal{W}_i)\right)$$

$$\mathrm{tm}(\mathcal{V}) = \mathrm{dis}(k) \cup \mathrm{evo}_k\left(\bigcap_{i=1}^{m} \mathrm{tm}(\mathcal{Z}_i)\right) \cup \mathrm{div}_k\left(\bigcap_{i=1}^{m} \mathrm{tm}(\mathcal{Z}_i)\right)$$

with $d(\mathcal{W}_1) = \cdots = d(\mathcal{W}_n) = d(\mathcal{Z}_1) = \cdots = d(\mathcal{Z}_m) < d(\mathcal{U})$. Hence, by induction hypothesis we have $\mathrm{tm}(\mathcal{W}_1) = \cdots = \mathrm{tm}(\mathcal{W}_n) = \mathrm{tm}(\mathcal{Z}_1) = \cdots = \mathrm{tm}(\mathcal{Z}_m)$, which implies $\mathrm{tm}(\mathcal{U}) = \mathrm{tm}(\mathcal{V})$. □

The following monotonicity result formalises the intuitive notion that, with the increase of $d(\mathcal{V})$, the possibilities for external objects to interfere with a path on the dependency graph may also increase.

Lemma 2. *Let* Π *be an MSD P system, let* $\mathcal{U}, \mathcal{V}$ *be two small configurations of* Π *such that, if both* $\mathcal{U}$ *and* $\mathcal{V}$ *contain two nested membranes, the internal ones have the same label, and assume that* $d(\mathcal{U}) \leq d(\mathcal{V})$*. Then* $\mathrm{tm}(\mathcal{U}) \subseteq \mathrm{tm}(\mathcal{V})$.

Proof. It suffices to show that $\mathrm{tm}(\mathcal{U}) \subseteq \mathrm{tm}(\mathcal{V})$ when $d(\mathcal{U}) = d(\mathcal{V}) - 1$. We prove it by induction on $d(\mathcal{U})$. If $d(\mathcal{U}) = 0$ or $d(\mathcal{U}) = 1$, then $\mathrm{tm}(\mathcal{U}) = \varnothing$, which is always a subset of $\mathrm{tm}(\mathcal{V})$.

If $d(\mathcal{U}) \geq 2$ then $d(\mathcal{V}) > 2$, and both $\mathrm{tm}(\mathcal{U})$ and $\mathrm{tm}(\mathcal{V})$ are computed using equation (2):

$$\mathrm{tm}(\mathcal{U}) = \mathrm{dis}(k) \cup \mathrm{evo}_k\left(\bigcap_{i=1}^{n} \mathrm{tm}(\mathcal{W}_i)\right) \cup \mathrm{div}_k\left(\bigcap_{i=1}^{n} \mathrm{tm}(\mathcal{W}_i)\right)$$

$$\mathrm{tm}(\mathcal{V}) = \mathrm{dis}(k) \cup \mathrm{evo}_k\left(\bigcap_{i=1}^{m} \mathrm{tm}(\mathcal{Z}_i)\right) \cup \mathrm{div}_k\left(\bigcap_{i=1}^{m} \mathrm{tm}(\mathcal{Z}_i)\right)$$

Since $d(\mathcal{Z}_1) = \cdots = d(\mathcal{Z}_m) = d(\mathcal{V}) - 1 = d(\mathcal{U})$, then by Lemma 1 we have $\mathrm{tm}(\mathcal{Z}_1) = \cdots = \mathrm{tm}(\mathcal{Z}_m) = \mathrm{tm}(\mathcal{U})$, and thus

$$\begin{aligned}\mathrm{tm}(\mathcal{V}) &= \mathrm{dis}(k) \cup \mathrm{evo}_k\left(\bigcap_{i=1}^{m} \mathrm{tm}(\mathcal{U})\right) \cup \mathrm{div}_k\left(\bigcap_{i=1}^{m} \mathrm{tm}(\mathcal{U})\right)\\ &= \mathrm{dis}(k) \cup \mathrm{evo}_k\left(\mathrm{tm}(\mathcal{U})\right) \cup \mathrm{div}_k\left(\mathrm{tm}(\mathcal{U})\right)\end{aligned}$$

Since $d(\mathcal{W}_1), \dots, d(\mathcal{W}_n) = d(\mathcal{U}) - 1$, we have $\mathrm{tm}(\mathcal{W}_1), \dots, \mathrm{tm}(\mathcal{W}_n) \subseteq \mathrm{tm}(\mathcal{U})$ by induction hypothesis, and thus $\bigcap_{i=1}^{n} \mathrm{tm}(\mathcal{W}_i) \subseteq \mathrm{tm}(\mathcal{U})$, which implies

$$\mathrm{tm}(\mathcal{V}) \supseteq \mathrm{dis}(k) \cup \mathrm{evo}_k \Big(\bigcap_{i=1}^{n} \mathrm{tm}(\mathcal{W}_i) \Big) \cup \mathrm{div}_k \Big(\bigcap_{i=1}^{n} \mathrm{tm}(\mathcal{W}_i) \Big) = \mathrm{tm}(\mathcal{U})$$

as needed. □

We call a configuration $\mathcal{C}$ of an MSD P system Π *untroubled* if it contains a vertex $\mathcal{V} \in V(\Pi)$ that is connected via a path to the vertex yes, and none of the troublemakers of $\mathcal{V}$. Notice, however, that a vertex can be contained multiple times in a given configuration, and each occurrence may or may not be subject to interference by its troublemakers. For instance, let $\mathcal{V} = \big[[a\ b]_k\big]_h$ with $\mathrm{tm}(\mathcal{V}) = \{c\}$ and $\mathcal{C} = \big[[a\ a\ b]_k\ [a\ b\ c]_k\big]_h$; the small configuration $\mathcal{V}$ occurs three times in $\mathcal{C}$ (twice in the left membrane k, and once in the right one), but only one occurrence may have interference, since the troublemaker c does not occur in the left membrane k. In order to formalise the notion of untroubled membrane, given a configuration of an MSD P system

$$\mathcal{C} = \big[[w_1]_{k_1}\ \cdots\ [w_n]_{k_n}\ w_0\big]_h\ w$$

we say that $\mathcal{L} \sqsubseteq \mathcal{C}$ is a *linear restriction of* $\mathcal{C}$ if either $\mathcal{L} = w$, or $\mathcal{L} = [w_0]_h$, or $\mathcal{L} = \big[[w_i]_{k_i}\big]_h$ for some $1 \le i \le n$. In a linear restriction $\mathcal{L}$, only one membrane contains objects, and thus the occurrences of a small configuration $\mathcal{V} \sqsubseteq \mathcal{L}$ are either all subject to interference from a troublemaker, or none of them is.

Definition 7. *We say that a configuration $\mathcal{C}$ of an MSD P system is* untroubled *if there exists a linear restriction $\mathcal{L} \sqsubseteq \mathcal{C}$ and a small configuration $\mathcal{V} \in Y(\Pi)$ such that $\mathcal{V} \sqsubseteq \mathcal{L}$ and no object $a \in \mathrm{tm}(\mathcal{V})$ appears in $\mathcal{L}$.*

The following results show that the property of being untroubled propagates forwards along a computation step and thus, recursively, all the way to the halting configuration.

Lemma 3. *Let Π be an MSD P system, let $\mathcal{C}$ be an untroubled configuration reachable from the initial configuration $\mathcal{C}_0$, and let $\mathcal{C} \to \mathcal{D}$. Then $\mathcal{D}$ is also untroubled.*

Proof. Since $\mathcal{C}$ is untroubled, there exists a linear restriction $\mathcal{L} \sqsubseteq \mathcal{C}$ and a small configuration $\mathcal{V} \sqsubseteq \mathcal{L}$ such that no object of $\mathrm{tm}(\mathcal{V})$ belongs to $\mathcal{L}$.

If $d(\mathcal{V}) \le 2$, then there exists a small configuration $\mathcal{W}$ with $d(\mathcal{W}) \le 1$ such that $\mathcal{V} \to \mathcal{V}' \sqsupseteq \mathcal{W}$ for some configuration $\mathcal{V}'$, otherwise $d(\mathcal{V})$ would be at least 3. But then $\mathrm{tm}(\mathcal{W}) = \varnothing$ and thus $\mathcal{D}$ is untroubled.

If $d(\mathcal{V}) \ge 3$, then $\mathcal{V} \to \mathcal{V}' \sqsupseteq \mathcal{W}$ for some $\mathcal{W}$ such that $d(\mathcal{W}) = d(\mathcal{V}) - 1 \ge 2$, and there exists a linear restriction $\mathcal{M} \sqsubseteq \mathcal{D}$ with $\mathcal{W} \sqsubseteq \mathcal{M}$. By contradiction suppose that, for all such $\mathcal{W}$, there exists $a \in \mathrm{tm}(\mathcal{W})$ with a belonging to $\mathcal{M}$.

This object a itself cannot appear in $\mathcal{L}$: since we have $\mathrm{tm}(\mathcal{W}) \subseteq \mathrm{tm}(\mathcal{V})$ by Lemma 2, we would have simultaneously $a \in \mathrm{tm}(\mathcal{V})$ and a in $\mathcal{L}$, contradicting our assumptions on $\mathcal{V}$. Hence, the object a must be generated by an object evolution or division rule triggered by an object b in $\mathcal{L}$.

If it is generated by an object evolution rule, then $b \in \mathrm{evo}_k\big(\mathrm{tm}(\mathcal{W})\big)$; since tm only depends on the distance d (Lemma 1), this means that we have $b \in \mathrm{evo}_k\big(\mathrm{tm}\big(\bigcap_{i=1}^n \mathcal{W}_i\big)\big)$, where the $\mathcal{W}_i$ are all small configurations, including $\mathcal{W}$, such that $(\mathcal{V}, \mathcal{W}_i) \in E(\mathit{\Pi})$ with $d(\mathcal{W}_i) = d(\mathcal{V}) - 1$. But then $b \in \mathrm{tm}(\mathcal{V})$ while simultaneously appearing in $\mathcal{L}$, once again contradicting the hypotheses on $\mathcal{V}$.

Suppose, instead, that the object a is generated by a division rule of the form $[b]_h \to [a]_h\ [c]_h$. We necessarily have $c \in \mathrm{tm}(\mathcal{W})$, because otherwise there would exist a linear restriction $\mathcal{M}' \sqsubseteq \mathcal{D}$ distinct from $\mathcal{M}$ with $\mathcal{M}'$ containing the other copy of $\mathcal{W}$ resulting from the division, but without any object in $\mathrm{tm}(\mathcal{W})$. But then we have both $a, c \in \mathrm{tm}(\mathcal{W})$, which implies that

$$b \in \mathrm{div}_k\big(\mathrm{tm}(\mathcal{W})\big) = \mathrm{div}_k\Big(\bigcap_{i=1}^n \mathrm{tm}(\mathcal{W}_i)\Big) \subseteq \mathrm{tm}(\mathcal{V})$$

once again contradicting the hypotheses on $\mathcal{V}$.

This shows that no object $a \in \mathrm{tm}(\mathcal{W})$ can belong to $\mathcal{M}$, and thus $\mathcal{D}$ is untroubled. □

Lemma 4. *Let $\mathit{\Pi}$ be an MSD P system with untroubled initial configuration $\mathcal{C}_0$. Then $\mathit{\Pi}$ accepts.*

Proof. Suppose that $\mathit{\Pi}$ has the halting computation $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \dots, \mathcal{C}_t)$ with untroubled $\mathcal{C}_0$. By Lemma 3, all other configurations of $\boldsymbol{\mathcal{C}}$, and in particular $\mathcal{C}_t$, are also untroubled. This means that there exist a linear restriction $\mathcal{L} \sqsubseteq \mathcal{C}_t$ and a small configuration $\mathcal{V} \in Y(\mathit{\Pi})$ such that $\mathcal{V} \sqsubseteq \mathcal{L}$ and no $a \in \mathrm{tm}(\mathcal{V})$ appears in $\mathcal{L}$. The only small configuration in $Y(\mathit{\Pi})$ where no rule is applicable is yes. This proves that the P system accepts. □

The property of being untroubled does not only propagate forwards, but also *backwards*, all the way to the initial configuration.

Lemma 5. *Let $\mathit{\Pi}$ be an MSD P system, let $\mathcal{C}$ be a configuration reachable from the initial configuration $\mathcal{C}_0$, and let $\mathcal{C} \to \mathcal{D}$ with $\mathcal{D}$ untroubled. Then $\mathcal{C}$ is also untroubled.*

Proof. Since $\mathcal{D}$ is untroubled, there exist a linear restriction $\mathcal{M} \sqsubseteq \mathcal{D}$ and a small configuration $\mathcal{W} \sqsubseteq \mathcal{M}$ such that no object $a \in \mathrm{tm}(\mathcal{W})$ appears in $\mathcal{M}$.

Since $\mathcal{W}$ contains at most two objects, it is generated by at most two rules applied in $\mathcal{C}$; this means that there exist a linear restriction $\mathcal{L} \sqsubseteq \mathcal{C}$ and a small configuration $\mathcal{V} \sqsubseteq \mathcal{L}$ such that $\mathcal{V} \to \mathcal{V}' \sqsupseteq \mathcal{W}$ for some configuration $\mathcal{V}'$. Let us consider the set $\mathrm{tm}(\mathcal{V})$. If $d(\mathcal{V}) \le 1$, this set is computed according to equation (1), and then $\mathrm{tm}(\mathcal{V}) = \varnothing$ and $\mathcal{C}$ is untroubled.

If $d(\mathcal{V}) \geq 2$, the set $\mathrm{tm}(\mathcal{V})$ is computed according to equation (2):

$$\mathrm{tm}(\mathcal{V}) = \mathrm{dis}(k) \cup \mathrm{evo}_k\Big(\bigcap_{i=1}^{n} \mathrm{tm}(\mathcal{W}_i)\Big) \cup \mathrm{div}_k\Big(\bigcap_{i=1}^{n} \mathrm{tm}(\mathcal{W}_i)\Big)$$

In this case we have $\mathcal{W} = \big[[a]_k\big]_h$ or $\mathcal{W} = \big[[a\ b]_k\big]_h$. Let us consider an object $a \in \mathrm{tm}(\mathcal{V})$. If $a \in \mathrm{dis}(k)$, then a does not appear in $\mathcal{L}$, because membrane k survived the transition $\mathcal{V} \to \mathcal{V}' \sqsupseteq \mathcal{W}$. If $a \in \mathrm{evo}_k\big(\bigcap_{i=1}^{n} \mathrm{tm}(\mathcal{W}_i)\big)$ appeared in $\mathcal{L}$, then it would evolve into an object of $\mathcal{M}$ belonging to $\bigcap_{i=1}^{n} \mathrm{tm}(\mathcal{W}_i) \subseteq \mathrm{tm}(\mathcal{W})$, which contradicts the hypotheses on $\mathcal{W}$ and is thus impossible. Similarly, if $a \in \mathrm{div}_k\big(\bigcap_{i=1}^{n} \mathrm{tm}(\mathcal{W}_i)\big)$ appeared in $\mathcal{L}$, then $\mathcal{M}$ would contain one of the two objects on the right-hand side of the division rule involving a, and that object would belong to $\mathrm{tm}(\mathcal{W})$; this contradicts the hypotheses on $\mathcal{W}$, and thus is also impossible. Hence, no object in $\mathrm{tm}(\mathcal{V})$ appears in $\mathcal{L}$, and thus $\mathcal{C}$ is untroubled. □

Lemma 6. *Let Π be an MSD P system with troubled initial configuration $\mathcal{C}_0$. Then Π rejects.*

Proof. Suppose that Π has the accepting computation $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_t)$. Then $\mathsf{yes} \sqsubseteq \mathcal{C}_t$ and $\mathrm{tm}(\mathsf{yes}) = \varnothing$, which means that $\mathcal{C}_t$ is untroubled. But we have $\mathcal{C}_0 \to \mathcal{C}_1 \to \cdots \to \mathcal{C}_t$, and by Lemma 5 all previous configurations, including $\mathcal{C}_0$, are untroubled, which contradicts our hypotheses. □

Hence, deciding whether an MSD P system accepts coincides with checking the troublemakers in its initial configuration.

Theorem 3. *An MSD P system accepts if and only if its initial configuration is untroubled.* □

If membrane dissolution rules are disallowed, the term $\mathrm{dis}(k)$ is always empty in equation (2), and thus $\mathrm{tm}(\mathcal{V}) = \varnothing$ for all small configurations $\mathcal{V}$; this implies that all configurations are untroubled (Definition 7). Furthermore, the small configurations of the form $\big[[a\ b]_k\big]_h$ can be ignored in favour of the smaller configurations $\big[[a]_k\big]_h$ and $\big[[b]_k\big]_h$, since a configuration with two objects is only required when the dissolution caused by one allows the other object to eventually evolve into yes (cases (5) and (6) of Theorem 1). This shows that, in the absence of dissolution, the acceptance condition of Theorem 3 corresponds to the existence of a path from a small configuration of the form $[a]_h$ or $\big[[a]_k\big]_h$, included in the initial configuration of the P system, to the small configuration yes. This is exactly the original acceptance condition for standard dependency graphs [3,5].

An analysis of the computational resources required in order to check the condition of Theorem 3 finally allows us to show that MSD P systems characterise the complexity class **P**.

Theorem 4. $\mathbf{DMC}_{\mathcal{D}}^{[\star]} = \mathbf{DPMC}_{\mathcal{D}}^{[\star]} = \mathbf{P}$*, where $\mathcal{D}$ is the class of MSD P systems and $[\star]$ denotes optional semi-uniformity.*

Proof. Since the following inclusions hold

$$\begin{tikzcd}[column sep=tiny, row sep=small]
 & \mathbf{DPMC}^\star_\mathcal{D} \arrow[dr, phantom, sloped, "\subseteq"] & \\
\mathbf{P} \subseteq \mathbf{DPMC}_\mathcal{D} \arrow[ur, phantom, sloped, "\subseteq"] \arrow[dr, phantom, sloped, "\supseteq"] & & \mathbf{DMC}^\star_\mathcal{D} \\
 & \mathbf{DMC}_\mathcal{D} \arrow[ur, phantom, sloped, "\subseteq"] &
\end{tikzcd}$$

it suffices to prove $\mathbf{DMC}^\star_\mathcal{D} \subseteq \mathbf{P}$. Let $\boldsymbol{\Pi}$ be a semi-uniform family of recogniser MSD P systems constructed in polynomial time by Turing machine H.

Given an input string $x \in \Sigma^\star$, simulate H on x in polynomial time in order to obtain the description of the P system Π_x.

The alphabet of Π_x and its set of rules have polynomial size, and thus the dependency graph $G(\Pi)$ can be constructed in polynomial time (see Gutiérrez-Naranjo et al. [5] for more details).

The set of vertices $Y(\Pi)$ connected to the vertex yes, and thus the subgraph of $G(\Pi)$ induced by them, can be computed in polynomial time by exploring the transposed dependency graph (i.e., with all edges reversed) starting from the vertex yes.

The value $d(\mathcal{V})$ can then be computed in polynomial time for each $\mathcal{V} \in Y(\Pi)$ by using an all-pairs shortest path algorithm on the subgraph induced by $Y(\Pi)$, and then choosing the distance from the closest vertex of the form $[a]_h$ or yes.

The troublemakers $\mathrm{tm}(\mathcal{V})$ are computed recursively in polynomial time using equations (1) and (2) on the subgraph induced by $Y(\Pi)$ and based on the distance function d.

Finally, for each of the (polynomially many) small configurations $\mathcal{V}$ and all (polynomially many) linear restrictions $\mathcal{L} \sqsubseteq \mathcal{C}_0$, where $\mathcal{C}_0$ is the initial configuration of Π, we can check whether $\mathcal{V} \sqsubseteq \mathcal{L}$ and simultaneously no object $a \in \mathrm{tm}(\mathcal{V})$ appears in $\mathcal{L}$. If this happens at least once, the input string x is accepted, and otherwise rejected. □

5 Conclusions and Open Problems

We have proved that families of monodirectional, deterministic, shallow P systems with active membranes without charges (MSD P systems) characterise the complexity class **P**, both with a polynomial-time uniformity and a polynomial-time semi-uniformity condition. This solves an open special case of the P conjecture and shows that dissolution does not always allow us to break the **P** barrier, even if object evolution, send-out, and membrane division rules are allowed.

In order to prove this result, we developed a generalisation of an important membrane computing tool, already exploited for several **P** upper bound results for P systems without charges: namely, dependency graphs. By proving that a constant number of objects govern the result of the computation, we were able to construct polynomial-size dependency graphs that include dissolving membranes. By checking a property of dependency graphs more complex than the original one, but still verifiable in polynomial time, we can establish the result of the computation of an MSD P system without resorting to expensive full simulations.

Even if we only focused on P systems of depth at most one in this paper, our approach should be straightforward to generalise to MSD P systems of any constant depth, as long as membrane division is only applicable to the initial elementary membranes, by tracking a constant number of objects larger than two. If elementary division rules are associated to membranes that only become elementary during the computation, there is the additional problem of establishing when this is the case for the membranes we are tracking (since "being elementary" depends on untracked membranes having dissolved). Here, we believe it is worth investigating a combination of our approach with the one proposed by Woods et al. [14], which deals correctly with dissolution and elementary division in P systems of arbitrary depth in the absence of other types of rules.

We further conjecture that it is possible to replace the determinism constraint with the standard confluence condition. This seems, however, to make the forwards and backwards propagation of the property of being untroubled harder to prove, due to the possibility of multiple computations.

We also remarked that several other generalisations of dependency graphs, for use with other variants of P systems, are possible. Although it is unclear whether this approach is powerful enough to solve the remaining open cases of the P conjecture, an interesting open problem is establishing which classes of P systems admit polynomial-size dependency graphs with polynomial-time computable accepting conditions. A related question is whether it is possible to find classes of P systems with larger-than-polynomial dependency graphs that do not require a complete exploration, and thus still allow their result to be established in polynomial time.

References

1. Alhazov, A., Freund, R.: 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, 5th International Workshop, WMC 2004. Lecture Notes in Computer Science, vol. 3365, pp. 146–160. Springer (2005)
2. Alhazov, A., Pérez-Jiménez, M.J.: Uniform solution to QSAT using polarizationless active membranes. In: Durand-Lose, J., Margenstern, M. (eds.) Machines, Computations, and Universality, 5th International Conference, MCU 2007, Lecture Notes in Computer Science, vol. 4664, pp. 122–133. Springer (2007)
3. Cordón-Franco, A., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Exploring computation trees associated with P systems. In: Mauri, G., Păun, Gh., Pérez-Jiménez, M.J., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, 5th International Workshop, WMC 2004. Lecture Notes in Computer Science, vol. 3365, pp. 278–286. Springer (2005)
4. Gazdag, Z., Kolonits, G.: Remarks on the computational power of some restricted variants of P systems with active membranes. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing, 17th International Conference, CMC 2016. Lecture Notes in Computer Science, vol. 10105, pp. 209–232. Springer (2017)

5. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Campero, F.J.: P systems with active membranes, without polarizations and without dissolution: A characterization of P. In: Calude, C.S., Dinneen, M.J., Păun, Gh., Pérez-Jímenez, M.J., Rozenberg, G. (eds.) Unconventional Computation, 4th International Conference, UC 2005. Lecture Notes in Computer Science, vol. 3699, pp. 105–116. Springer (2005)
6. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Campero, F.J.: On the power of dissolution in P systems with active membranes. In: Freund, R., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, 6th International Workshop, WMC 2005. Lecture Notes in Computer Science, vol. 3850, pp. 224–240. Springer (2006)
7. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Simulating elementary active membranes, with an application to the P conjecture. In: Gheorghe, M., Rozenberg, G., Sosík, P., Zandron, C. (eds.) Membrane Computing, 15th International Conference, CMC 2014, Lecture Notes in Computer Science, vol. 8961, pp. 284–299. Springer (2014)
8. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Monodirectional P systems. Natural Computing 15(4), 551–564 (2016)
9. Murphy, N., Woods, D.: Active membrane systems without charges and using only symmetric elementary division characterise P. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, 8th International Workshop, WMC 2007. Lecture Notes in Computer Science, vol. 4860, pp. 367–384 (2007)
10. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Natural Computing 10(1), 613–632 (2011)
11. Păun, Gh.: Further twenty six open problems in membrane computing. In: Gutíerrez-Naranjo, M.A., Riscos-Nuñez, A., Romero-Campero, F.J., Sburlan, D. (eds.) Proceedings of the Third Brainstorming Week on Membrane Computing. pp. 249–262. Fénix Editora (2005)
12. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2(3), 265–284 (2003)
13. Sosík, P., Rodríguez-Patón, A.: Membrane computing and complexity theory: A characterization of PSPACE. Journal of Computer and System Sciences 73(1), 137–152 (2007)
14. Woods, D., Murphy, N., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Membrane dissolution and division in P. In: Calude, C.S., Félix Costa, J., Dershowitz, N., Freire, E., Rozenberg, G. (eds.) Unconventional Computation, 8th International Conference, UC 2009, Lecture Notes in Computer Science, vol. 5715. Springer (2009)

An Improved Partitioning Around Medoids Algorithm Based on Time-Free Cell-Like P Systems with Multi-Promoters and Multi-Inhibitors

Xiyu Liu, Yuzhen Zhao [⋆], Wenping Wang, and Wenxing Sun

School of Management Science and Engineering, Shandong Normal University, Jinan, 250014, China
sdxyliu@163.com
zhaoyuzhen_happy@126.com
64759428@qq.com
373253360@qq.com

Abstract. Partitioning Around Medoids algorithm (PAM, for short) is a typical clustering algorithm based on the K-medoids method, which is based on the representative objects in the database and therefore is robust to noises and outliers. To improve the time efficiency of PAM, a new variety of cell-like P system, called time-free cell-like P systems with multi-promoters and multi-inhibitors, is proposed which is used to parallelize the operation, and an improved PAM algorithm based on the new variant of P system (TFP-PAM, for short) is proposed in this study. In each iterative process of TFP-PAM, only the q-nearest neighbors of each representative object (medoid) of a cluster are considered as the candidate medoids instead of each non-medoid in each iterative process of the original PAM, which decreases the number of the candidate medoids. Through theoretical analysis, the TFP-PAM decreases the time complexity. Evidenced by experiments, the clustering quality of TFP-PAM is as good as the original PAM.

Keywords: PAM; K-medoids; P system; Time-free; Cell-like P system; Multi-promoter; Multi-inhebitor

1 Introduction

Information plays an important role in each field in modern society. It is an important problem to analyze data and extract useful information from the huge amounts of data. Clustering analysis is a class of important data analysis methods, which can reveal the relationship between data objects and data objects, data objects and data features, data features and data features. K-medoids

[⋆] Project supported by National Natural Science Foundation of China (No. 61472231, 61502283, 61640201, 61602282) and Natural Science Foundation of Shandong Province (No. ZR2016AQ21).

method is a typical clustering method, which is based on the representative objects in the database and therefore is robust to noises and outliers [1]. The Partitioning Around Medoids algorithm (PAM, for short) is one of the popular K-medoids clustering algorithms. While, for big database, the scalability of PAM is not good. For dealing with big database, CLARA algorithm and CLARANS algorithm were proposed [1, 2].

In order to further improve the time efficiency of PAM, a novel distributed and parallel computing model called the P system is introduced. P systems are new bio-inspired computing models of membrane computing, which focuses on abstracting computing ideas from the study of biological cells, particularly of cellular membranes [3, 19]. P systems are powerful computing devices, being able to do what Turing machines can do [5–7], and have been applied to many fields.

The applications of P systems are based on two types of membrane algorithms, the coupled membrane algorithm and the direct membrane algorithm. The coupled membrane algorithm combines the traditional algorithm with some structural characters of P systems, such as dividing the whole system into several relatively independent computing units, where the computing units can communicate with each other, the computing units can be dynamically rebuilt, and rules can be executed in parallel [8–12]. The direct membrane algorithm designs the algorithm based on the structure, the objects and the rules of P systems directly [13–17]. The final goal of membrane computing is to build bio-computers and the direct membrane algorithm can be transplanted to the bio-computers directly, which is more meaningful from this perspective. However, the direct membrane algorithm needs to transform the whole traditional algorithm into P system, which is complex and difficult. Up to date, a few simple studies on the direct membrane algorithm focus on the arithmetic operations, the logic operations, the generation of graphic language and clustering [13–17].

In this study, a variant of cell-like P systems with promoters and inhibitors called time-free cell-like P systems with multi-promoters and multi-inhibitors (TFMCPI-P, for short) is proposed. And then a novel improved PAM algorithm based on the TFMCPI-P (TFP-PAM, for short) is proposed using the parallel mechanism in P systems. The information communication among different computing units in TFP-PAM is implemented through the exchange of materials between membranes. Specifically, the best medoids of all clusters are searched in parallel, regulated by a set of promoters and inhibitors. For a database which needs to be clustered into K clusters, $K + 3$ membranes are used in the algorithm, where one membrane is used to obtain the q-nearest neighbors of each data object in the database, K membranes are used to store the data objects of each cluster, and to find the best representative objects (medoids), one membrane is used to store the results, and one membrane is used as the container which contains all these membranes. The time complexities of TFP-PAM is compared with those of other K-medoids methods to show the time savings of the proposed algorithm.

The contributions of this study are two fields. From the viewpoint of data mining, the new bio-inspired technique is introduced into PAM to improve the efficiency of the algorithm. P systems are natural distributed parallel computing devices which can improve the time efficiency in computation. Besides the hardware and software implementations, P systems can be implemented by biological methods. The computing resources needed are only several cells, which can decrease the computing resource requirements. From the viewpoint of P systems, a new variety of cell-like P system, called time-free cell-like P systems with multi-promoters and multi-inhibitors, is proposed. Furthermore, the application areas of the new bio-inspired devices P systems are extended to PAM. The applications based on the direct membrane algorithms are limited. This study provides a new application of P systems, which expands the application areas of the direct membrane algorithms.

The paper is organized as follows. Section 2 introduces the improved PAM based on the q-nearest neighbors. The TFMCPI-P system is proposed in section 3. The TFP-PAM algorithm using the parallel mechanism of the TFMCPI-P system is developed in Section 4. In Section 5, the computational process of the TFP-PAM is analysed, and the time complexities between TFP-PAM and other K-medoids methods are compared. Computational experiments using two databases show the performance of the proposed algorithm in section 6. Conclusions are given in Section 7.

2 The Improved PAM Algorithm Based on the q-Nearest Neighbors

PAM algorithm is a classical K-medoids clustering algorithm improved by the K-means algorithm. It is more robust to noises and outliers as compared to K-means algorithm [1].

For a data set $X = \{x_1, x_2, ..., x_n\}$ of n data, PAM can cluster it into K clusters $C = \{C_1, C_2, ..., C_K\}$. A objective function is used to measure the quality of the clustering as follows:

$$E = \sum_{j=1}^{K} \sum_{x_i \in C_j} |x_i - o_j| \tag{1}$$

where, E is the sum of the absolute error, x_i is the object in the data set, and o_j is the medoid of cluster C_j. That is to say, $|x_i - o_j|$ is the Euclidean distance between the objects x_i and o_j.

The final clusters have the following properties:

1. $C_i \neq \emptyset, 1 \leq i \leq K$;
2. $\cup_{i=1}^{K} C_i = X$;
3. $C_i \cap C_j = \emptyset, i \neq j, 1 \leq i, j \leq K$.

The steps of PAM are as follows:

1. Select K data points as the initial medoids randomly.
2. Assign all non-medoids data points to the cluster with the nearest medoid evaluated by the Euclidean distance measures.
3. Select a non-medoid point o'_j randomly as a candidate medoid.
4. Compute the absolute-error criterion E' when o_j is instead of o'_j.
5. Replace the medoid o_j by o'_j if $E' < E$.
6. Repeat steps 2 to 5 until all medoids are not changed.

The conventional PAM selects a non-medoid object o'_j randomly as the candidate medoid, and the value of E and E' of the whole database are computed. For big database, the computational time is long which limits the application scape of PAM. In this study, only the q-nearest neighbors of the medoid can be chosen as the candidate medoid. Furthermore, when the candidate medoid o'_j and the conventional medoid o_j are compared, only the data points within the current cluster C_j are considered. That is, only the value of $\sum\limits_{x_i \in C_j} |x_i, o'_j|$ and $\sum\limits_{x_i \in C_j} |x_i, o_j|$ are compared. These two changes can improve the time efficiency.

The steps of the improved PAM based on the q-nearest neighbors are as follows:

1. Select K data objects as the initial medoids randomly.
2. Assign all non-medoids data objects to the cluster with the nearest medoid.
3. Select a non-medoid object o' from the q-nearest neighbors of the original medoid randomly as a candidate medoid.
4. Compute $\sum\limits_{x_i \in C_j} |x_i, o'_j|$ and $\sum\limits_{x_i \in C_j} |x_i, o_j|$.
5. Replace the medoid by o'_j if $\sum\limits_{x_i \in C_j} |x_i, o'_j| < \sum\limits_{x_i \in C_j} |x_i, o_j|$.
6. Repeat steps 2 to 5 until all medoids are not changed.

3 Time-Free Cell-Like P Systems with Multi-Promoters and Multi-Inhibitors

Membrane computing is a new branch of natural computing, which abstracts computing ideas from the construct and the functions of cells or tissues. In the nature, each organelle membrane or cell membrane works as a relatively independent computing unit. The amount and the types of materials in each organelle or cell change through chemical reactions. Materials can flow between different organelle or cell membranes to transport information. Reactions in different organelles or cells take place in parallel, while reactions in the same organelle or cell take place also in parallel. These biological processes are abstracted as the computational processes of membrane computing. The internal parallel feature makes membrane computing a powerful computing method which has been proven to be equivalent to Turing machines.

Cell-like P systems with promoters and inhibitors are abstracted based on the structure and function of the living cell, which have three main components, the membrane structure, multisets of objects evolving in a synchronous maximally parallel manner, and evolution rules. Objects in P system evolve in a maximum parallel mechanism, regulated by promoters and inhibitors [19]. Rules regulate the ways objects evolve to new objects and the ways objects in different membranes communicate with each other. Rules are executed in non-deterministic maximally parallel manner in each cell. That is, at any step, if more than one rule can be executed but the objects in the membrane can only support some of them, then a maximal number of rules will be executed. A rule can be executed when all promoters are present, and cannot be executed when all inhibitors are present. Each P system contains a global clock as the timer, and the execution time of one rule is set to a time unit.

In this study, the amount of the promoters and the inhibitors are extended, that is to say, a certain rule can have multi-promoters and multi-inhibitors, the meaning of each promoter and inhibitor is the same as before, while a rule can be executed when any of the promoters are present, and cannot be executed when any of the inhibitors are present. And in practice, the execution time of different biochemical reactions is affected by external uncontrollable factors, which means the execution time of different rules is different. By adding the mapping $e : R_1 \bigcup R_2 \bigcup \cdots \bigcup R_m \rightarrow \mathbb{N} \backslash \{0\}$ from the set of rules to the set of natural numbers which represents the execution time of rules, a timed cell-like P systems with multi-promoters and multi-inhibitors (TMCPI-P systems, for short) is constructed. And a TFMCPI-P system is called a time-free cell-like P systems with multi-promoters and multi-inhibitors (TFMCPI-P systems, for short) if and only if every TFMCPI-P system in the set e produces the same set of vectors of natural numbers.

The formal description of the TFMCPI-P system is as follows.

$$\mathit{\Pi}(e) = (O, \mu, w_1, w_2, \ldots, w_m, R_1, \ldots, R_m, \rho, i_{in}, i_{out}, e), \text{where,}$$

- O is the alphabet which includes all objects of the system.
- μ is the membrane structure.
- w_i is the initial objects in membrane i, object λ shows that there is no object in membrane i.
- R_i is the set of rules in membrane i with the form of $u_\alpha \rightarrow v$, where u is a string composed of objects in O and v is a string over $\{a_{here}, a_{out}, a_{in_j} | a \in O, 1 \leq j \leq t\}$. ($a_{here}$ means object a remains in membrane i in which $here$ can be omitted; a_{out} means object a goes into the outer layer membrane; and a_{in_j} means object a goes into the inner layer membrane j.) α is a string over $\{z, \neg z'\}$ represents the multi-promoters and multi-inhibitors. (Each z_i is a promoter and each $\neg z'$ is an inhibitor. A rule can be executed when any promoter z appears and cannot be executed when any inhibitor z'_i appears.) Different promoters or inhibitors are distinguished by comma.
- ρ defines the partial ordering relationship of the rules, i.e., rules with higher orders are executed with higher priority.
- i_{in} is the membrane where the objects are put into.
- i_{out} is the membrane where the computation result is placed.

– e is a mapping from the set of rules to the set of natural numbers which represents the execution time of rules $e : R_1 \bigcup R_2 \bigcup \cdots \bigcup R_m \rightarrow \mathbb{N}\backslash\{0\}$. The execution time of rule r is $e(r)$. If rule r is executed at step t, the execution process is over at step $t + e(r)$.

The computation halts if no rule can be executed in the whole system. The computational results are represented by the types and numbers of specified objects in a specified membrane. Because objects in a P system evolve in maximally parallel, regulated by promoters and inhibitors, the systems compute very efficiently. Păun [19] provided more details about P systems.

Note that in time-free P systems, a step when some rules begin to be executed is called a RS step. When the time complexity of a time-free P system is computed, only the number of RS steps in the computational process is considered.

4 The TFP-PAM Algorithm

In this section, an improved PAM algorithm based on time-free cell-like P systems with promoters and inhibitors is proposed, where promoters and inhibitors are utilized to regulate parallelism of objects evolution. The obtained algorithm is shortly called TFP-PAM.

Before introducing TFP-PAM, the distance matrix W_{nn} is defined as follows:

$$W_{nn} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ & \cdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{pmatrix}. \tag{2}$$

where w_{ij} is the Euclidean distance between the objects x_i and x_j.

The membrane structure used as the framework for TFP-PAM is shown in Figure 1.

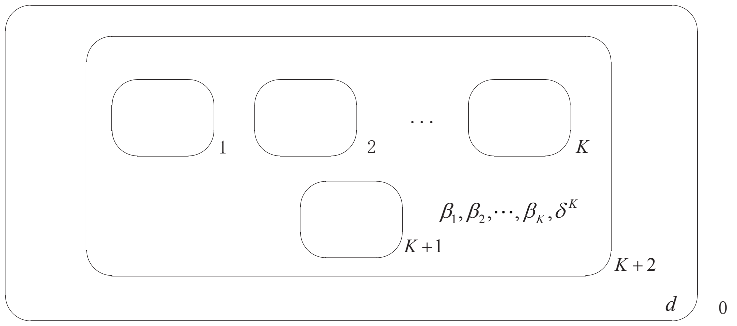


Fig. 1. Membrane structure for TFP-PAM

The time-free cell-like P system with promoters and inhibitors for TFP-PAM is as follows.

$$\Pi(e) = (O, \mu, w_0, w_1, \cdots, w_{K+2}, R_0, R_1, \cdots, R_{K+2}, \rho, i_{in}, i_{out}, e), \text{where,}$$

- $O = \{a_i, U'_{ij}, U_{ij}, c, c_i, c_{ij}, c'_{ij}, d, d', \xi, \beta_k, A_{ki}, A'_{ki}, B_{ki}, \delta, \delta', D_{ij}, \psi, \theta, \alpha, a_{ij}, O_i, e_i, e, g, \tau, \delta_i, \zeta_i, s, s', \eta, \sigma, b_i\}$;
- $\mu = [[[]_1[]_2 \cdots []_{K+1}]_{K+2}]_0$;
- $w_i = \lambda\ (i = 0, 1, 2, \cdots, K+1)$,
  $w_{K+2} = \{\beta_1, \beta_2, \cdots, \beta_K, \delta^K\}$;
- $\rho = \{r_{ij_1} > r_{ij_2} | i = 0, 1, \cdots, K, K+2, j_1 < j_2\}$;
- $i_{in} = 0$;
- $i_{out} = K+1$;
- $e : R_1 \bigcup R_2 \bigcup \cdots \bigcup R_m \to \mathbb{N} \backslash \{0\}$.
- R_i is the set of rules in membrane i.

R_0:

$$\begin{cases} r_{0,1} = \{U'_{ij} \to D_{ij}U_{ij}\}(i, j = 1, 2, \cdots, n) \\ r_{0,2} = \{c_{i\neg(D_{ij}, c_{ij})} \to c_{ij}d'\}(i, j = 1, 2, \cdots, n) \\ r_{0,3} = \{(D_{i1}D_{i2}...D_{in})_c \to \lambda\}(i = 1, 2, \cdots, n) \\ r_{0,4} = \{(D_{ij_1}D_{ij_2}...D_{ij_{(n-1)}})_{c^2} \to \lambda\}(i, j_1, j_2, ...j_{(n-1)} = 1, 2, \cdots, n) \\ r_{0,5} = \{(D_{ij_1}D_{ij_2}...D_{ij_{(n-2)}})_{c^3} \to \lambda\}(i, j_1, j_2, ...j_{(n-2)} = 1, 2, \cdots, n) \\ ... \\ r_{0,n+1} = \{(D_{ij_1}D_{ij_2})_{c^{n-1}} \to \lambda\}(i, j_1, j_2 = 1, 2, \cdots, n) \\ r_{0,n+2} = \{dd' \to \lambda\} \\ r_{0,n+3} = \{(a_i)_{\neg d} \to (a_i)_{in_{K+2}}\} \bigcup \{(U_{ij})_{\neg d} \to {U'}_{ij\, in_{K+2}}\} \\ \qquad\qquad \bigcup \{(c_{ij})_{\neg d} \to ({c'}_{ij})_{in_{K+2}}\}(i, j = 1, 2, \cdots, n) \end{cases}$$

R_{K+2}:

$$\begin{cases} r_{K+2,1} = \{{U'}_{ij} \to U_{ij}(U_{ij})_{in_{1,2,\cdots,K}}\} \bigcup \{{c'}_{ij} \to c_{ij}(c_{ij})_{in_{1,2,\cdots,K}}\} \\ \qquad\qquad (i, j = 1, 2, \cdots, n) \\ r_{K+2,2} = \{a_i\beta_k \to A_{ki}(A'_{ki})_{in_k}\}(i = 1, 2, \cdots, n; k = 1, 2, \cdots, K) \\ r_{K+2,3} = \{a_{i_{\delta^K A_{1i_1} A_{2i_2} \cdots A_{Ki_K} U_{ii_1}^{W_{ii_1}} U_{ii_2}^{W_{ii_2}} \cdots U_{ii_K}^{W_{ii_K}} \neg D_{i1}^{W_{ii_1}} D_{i2}^{W_{ii_2}} \cdots D_{iK}^{W_{ii_K}}}} \to \\ \qquad\qquad b_i D_{i1}^{W_{ii_1}} D_{i2}^{W_{ii_2}} \cdots D_{iK}^{W_{ii_K}} \delta'\}(i, i_1, i_2, \cdots, i_K = 1, 2, \cdots, n) \\ r_{K+2,4} = \{{\delta'}^{n-K}\delta^K \to \lambda\} \\ r_{K+2,5} = \{b_{i\neg D_{ik}} \to \xi a_{i\ in_k}\}(i = 1, 2, \cdots, n; k = 1, 2, \cdots K) \\ r_{K+2,6} = \{D_{i1}D_{i2} \cdots D_{iK} \to \lambda\}(i = 1, 2, \cdots, n) \\ r_{K+2,7} = \{(D_{ik})_\xi \to \lambda\}(i = 1, 2, \cdots, n; k = 1, 2, \cdots, K) \\ r_{K+2,8} = \{A_{1i_1} \cdots A_{Ki_K}\xi^{n-K}\psi^i g^j \to \delta^K(\theta)_{in_{1,2,\cdots,K}}\} \\ \qquad\qquad \bigcup \{A_{1i_1} \cdots A_{Ki_K}\xi^{n-K}\psi^K \to \delta^K(\alpha)_{in_{1,2,\cdots,K}}\} \\ \qquad\qquad (i_1, i_2, \cdots, i_K = 1, 2, \cdots, n; i = 0, 1, \cdots K-1; j = 1, 2, \cdots, K) \\ r_{K+2,9} = \{a_{ik} \to a_{ik\ in_{K+1}}\}(i = 1, 2, \cdots, n; k = 1, 2, \cdots K) \end{cases}$$

$R_k, k = 1, 2, \cdots, K$:

$$\begin{cases} r_{k,1} = \{(a_i)_\alpha \rightarrow a_{ik\ out}\}(i = 1, 2, \cdots, n) \\ r_{k,2} = \{(\theta A_{ki} B_{ki})_{\neg a_j} \rightarrow A_{ki}(\psi A_{ki}\delta)_{out}\#\}(i, j = 1, 2, \cdots, n) \\ r_{k,3} = \{A'_{ki} \rightarrow A_{ki} B_{ki}\}(i = 1, 2, \cdots, n) \\ r_{k,4} = \{(\theta a_j c_{ij})_{B_{ki}} \rightarrow O_j c'_{ij}\} \bigcup \{(\theta c_{ij})_{B_{ki} \neg a_j} \rightarrow \theta c'_{ij}\}(i, j = 1, 2, \cdots, n) \\ r_{k,5} = \{(a_t)_{O_j A_{ki} U_{jt}^{W_{jt}} U_{it}^{W_{it}} \neg e_j} \rightarrow e_j a_t s^{W_{jt}} {s'}^{W_{it}}\}(i, j, t = 1, 2, \cdots, n) \\ r_{k,6} = \{ss' \rightarrow \lambda\} \\ r_{k,7} = \{({s'}^t O_j A_{ki})_{\neg s} \rightarrow \theta a_i A_{kj} \eta\} \bigcup \{(s^t O_j A_{ki})_{\neg s'} \rightarrow \theta a_j A_{ki} \sigma\} \\ \qquad (i, j = 1, 2, \cdots, n, t = 1, 2, \cdots, |maxf_{ij} - minf_{ij}|) \\ r_{k,8} = \{(\theta)_{B_{ki} \neg c_{ij}} \rightarrow \tau\}(i, j = 1, 2, \cdots, n) \\ r_{k,9} = \{(c'_{ij})_\tau \rightarrow c_{ij}\} \bigcup \{(e_i)_\tau \rightarrow \lambda\}(i, j = 1, 2, \cdots, n) \\ r_{k,10} = \{(\eta^i \sigma^j \tau) \rightarrow dg_{out}\}(i = 1, 2, \cdots, n; j = 0, 1, \cdots, n) \\ \qquad \bigcup \{(\sigma^j \tau)_{\neg \eta^i} \rightarrow d\psi_{out}\}(i, j = 1, 2, \cdots, n) \\ r_{k,11} = \{(a_i)_d \rightarrow e(a_i)_{out}\}(i = 1, 2, \cdots, n) \\ r_{k,12} = \{de^j A_{ki} B_{ki} \rightarrow A_{ki}(\delta A_{ki})_{out}\#\}(i, j = 1, 2, \cdots, n) \end{cases}$$

5 The Computational Process and Complexity Analysis

The computational process is as follows.

The generation of the q-nearest neighbors of each object in the database

Objects a_i, ${U'_{ij}}^{W_{ij}}$, c^q, c_i^q and d^{nq} are put into membrane 0 to start the computational process. Object a_i means the $i-th$ data in the database; object ${U'_{ij}}^{W_{ij}}$ means the dissimilarity between a_i and a_j is W_{ij}, in other words, the dissimilarity is represented by the number of objects U'_{ij}; objects c^q are auxiliary objects which means the q-nearest neighbors of each object are considered; object c_i^q means the q-nearest neighbors of object a_i are still needed to be found. In the computational process, the number of objects c_i decreases by one when one more nearest neighbor of a_i is found; objects d^{nq} means nq nearest neighbors of all objects a_i need to be obtained.

Rule $r_{0,1}$ is executed to generate objects D_{ij} and U_{ij}. Rule $r_{0,3}$ is executed to dissolve the same number of objects $D_{i1}D_{i2}...D_{in}$. The number of objects U_{ij_1} which are all dissolved by rule $r_{0,3}$ is the minimum one among all objects $D_{i1}D_{i2}...D_{in}$, which means object a_{j_1} is the nearest neighbor of object a_i. Rule $r_{0,2}$ is executed to generate an object c_{ij_1} to store the result. Then, rule $r_{0,4}$ is executed to dissolve the same number of objects $D_{i1}D_{i2}...D_{ij_{(n-1)}}$. The number of objects U_{ij_2} which are all dissolved by rule $r_{0,4}$ is the minimum one among all objects $D_{i1}D_{i2}...D_{ij_{(n-1)}}$, which means object a_{j_2} is the second-nearest neighbor of object a_i. Rule $r_{0,2}$ is executed to generate an object c_{ij_2} to store the result. The above process will continue until rule $r_{0,q+2}$ is executed to dissolve the same number of objects $D_{i1}D_{i2}...D_{i(n-q+1)}$. The number of objects U_{ij_q} which are all dissolved by rule $r_{0,q+2}$ is the minimum one among all objects $D_{i1}D_{i2}...D_{i(n-q+1)}$, which means object a_{j_q} is the $(q-th)$-nearest neighbor of object a_i. Rule $r_{0,2}$ is executed to generate an object c_{ij_q} to store the result.

When all q-nearest neighbors of objects $a_1, a_2, \cdots, a_n$ have been obtained, rule $r_{0,n+3}$ is executed to send copies of a_i, U'_{ij} and c'_{ij} to membrane $K+2$.

The distribution of objects in the database to the nearest cluster

Rule $r_{K+2,1}$ is executed to generate objects $U_{ij}^{W_{ij}}$ and c_{ij}, one copy of the above objects are stored in membrane $K+2$, and one copy of the above objects are sent to membranes $1, 2, \cdots, K$. At the same time, rule $r_{K+2,2}$ is executed. Objects $\beta_1, \beta_2, \cdots, \beta_K$ are in membrane $K+2$ showing that K medoids are needed in the initial state. Rule $r_{K+2,2}$ chooses K medoids A_{ki} randomly, one copy of objects A_{ki} are stored in membrane $K+2$, and one copy of objects A_{ki} are changed to A'_{ki} and are sent to membranes $1, 2, \cdots, K$. Then, rule $r_{K+2,3}$ is executed to generate objects $D_{ik}^{W_{ii_k}}$, which mean the dissimilarity between object a_i and the $k-th$ medoid is W_{ii_k}. Rule $r_{K+2,6}$ is executed to dissolve the same number of objects $D_{i1}D_{i2}\cdots D_{iK}$. The number of objects D_{ik} which are all dissolved by rule $r_{K+2,6}$ is the minimum one among all objects $D_{i1}D_{i2}\cdots D_{iK}$, which means the dissimilarity between a_i and the $k-th$ cluster is the smallest one, object a_i is sent to membrane k. Rules $r_{K+2,4}$ and $r_{K+2,7}$ are executed to dissolve the useless objects. Rule $r_{K+2,8}$ is executed to send an object θ to each of membranes $1, 2, \cdots, K$ to start the computational process in membranes $1, 2, \cdots, K$.

When all medoids in the K clusters are not changed, which means the final clusters are formed, an object α is sent to these K membranes by rule $r_{K+2,8}$. Rule $r_{K+2,9}$ is executed to send objects a_{ik} to membrane $K+1$, which means a_i belongs to the $k-th$ cluster.

The redetermine of the best medoid in the k-th cluster

The best medoid among all the q-nearest neighbors and the current medoid A_{jh} is found in the membrane.

Rule $r_{k,3}$ is executed to generate objects A_{ki} and B_{ki}, where A_{ki} is used to store the medoid, and B_{ki} is used to store the original medoid in this iteration.

If no object belongs to this cluster except for the medoid, rule $r_{k,2}$ is executed, objects ψ, A_{ki} and δ are sent out showing that the medoid of this cluster is not changed in this iteration.

If there are other objects belonging to this cluster except for the medoid, rule $r_{k,4}$ is executed, the q-nearest neighbors of the original medoid are checked randomly. If a certain neighbor is not in this cluster, object c_{ij} is changed to c'_{ij} to show this neighbor a_j has been checked. If a certain neighor is in this cluster, a_j is changed to O_j to show that a_j is set as the candidate medoid. Rule $r_{k,5}$ is executed to calculate dissimilarity sum of the original medoid and the candidate medoid. The dissimilarity sum value of the candidate medoid is stored by the number of object s. Similarly, the dissimilarity sum value of the original medoid is stored by the number of object s'. Rule $r_{k,6}$ is executed to dissolve the same number of objects s and s'. Rule $r_{k,7}$ is executed to compare the two medoids. If s' is still in the membrane, which means the candidate medoid can reduce the dissimilarity sum, object a_j is set as the new medoid. If s is still in the membrane, which means the candidate medoid cannot reduce the dissimilarity sum, the medoid is still the old one. Rules $r_{k,4}$ to $r_{k,7}$ are executed until all the q-nearest neighbors of the original medoid are checked. If the medoid in this membrane is changed, an object g is sent out, otherwise, an object ψ is sent out by rule $r_{k,10}$. Lastly, rules $r_{k,11}$ and $r_{k,12}$ are executed to renew the objects in this membrane to the initial state, and put the object A_{jp} out. Object # is generated to stop the computational process.

When the clustering process is over, an object α is sent to this membrane. Rule $r_{k,1}$ is executed to send out an object a_{ik} showing that a_i belongs to the $k-th$ cluster.

The time complexity for each iteration is $1+1+1+1+1+1+1++1+(1+1+1+1+1)*q+1+1+1+1+1 = 5q+13 = O(1)$.

Some comparison results between TFP-PAM and the original PAM as well as some other K medoids algorithms are shown in Table 1, where s is the sample size.

Table 1. Time complexities for each iteration of some K medoids algorithms

Algorithm	Time complexity
PAM [1]	$O(K(n-K))^2$
CLARA [1]	$O(Ks^2+K(n-K))$
CLARANS [2]	$O(n^2)$
TFP-PAM	$O(1)$

6 Experiments and Analysis

Iris

The Iris database of UC Irvine Machine Learning Repository is used as an experiment [18]. This database contains 150 records. The 150 records are numbered from 1 to 150 following the order. Each record contains four Iris property values and the corresponding Iris species. All records are divided into three species, data from 1 to 50, data from 51 to 100 and data from 101 to 150, respectively.

Let $K = 3, q = 50$, the original PAM and TFP-PAM are used for 100 times respectively to cluster this database. The clustering accuracies and the iteration times are shown in Fig.2, Fig.3, Fig.4 and Fig.5.

We can see from the figures, the clustering accuracies of the two algorithms are similar, the average clustering accuracy of PAM is 70.36%, and the average clustering accuracy of TFP-PAM is 70.49%. As to the iteration times, the iteration times of the original PAM is 2 all the time, while the iteration times of TFP-PAM are fluctuant between 2 and 5, with an average value of 2.78. The average iteration times of TFP-PAM is a little higher, however, the time for each iteration in TFP-PAM is much lower than PAM.

The value of q is set 1 to 149 to study the influence of the parameter q. For each q value, TFP-PAM is used 10 times. The clustering accuracies and the iteration times are shown in Fig.6 and Fig.7.

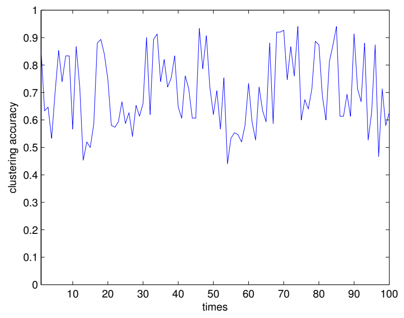


Fig. 2. The clustering accuracies of the original PAM

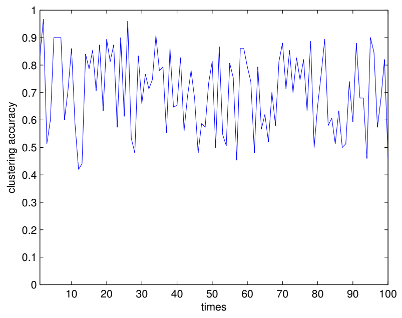


Fig. 3. The clustering accuracies of TFP-PAM

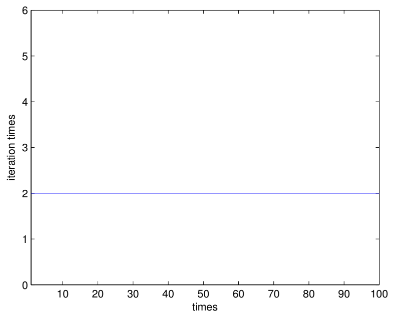


Fig. 4. The iteration times of the original PAM

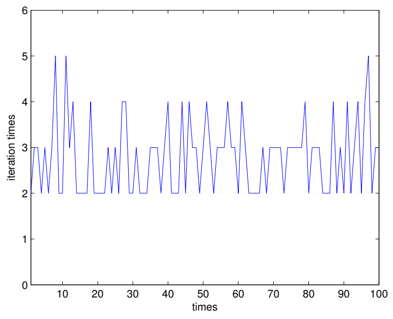


Fig. 5. The iteration times of TFP-PAM

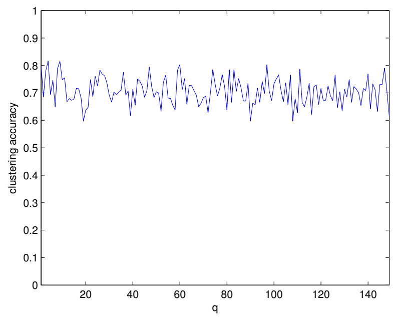


Fig. 6. The clustering accuracies of TFP-PAM with different q values

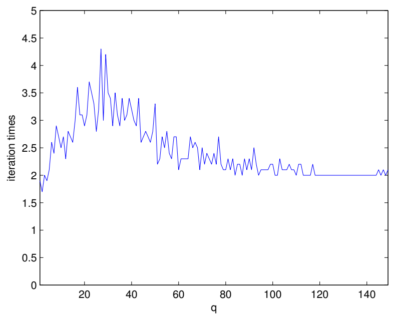


Fig. 7. The clustering accuracies of TFP-PAM with different q values

We can see from the figures, the clustering accuracy is almost uninfluenced by the q values. While the iteration times rise at first and go down latter, and finally tend to be stable. The largest iteration time is got at $q = 27$. That is to say, even if we only consider very few nearest neighbors in TFP-PAM, the clustering accuracy is still stable.

short text

45 chinese news titles are selected as the database, and each title is treated as a data object. The 45 data objects are numbered from 1 to 45 following the order, where 24 titles belong to class "education", and 21 titles belong to class "entertainment".

Firstly, the database is pretreated. Each data object is segmented into several words using a given software, and then only part of the words are reserved according to their parts of speech. Next, the stop words are deleted. All 172 words in the pretreated database form the word dictionary. The word frequency row vector of each data object is computed which shows the frequency of occurrence of each word in the word dictionary. Each data object is represented by the $tfidf$ values of each element if the word frequent row vector. The ten words of a data object are selected as the keywords of this data object which $tfidf$ values are the ten largest ones, so only the values of the ten keywords are reserved, and other elements values in a data object are set "0". Therefore, each object in this database is a row vector with 172 elements.

Let $K = 2, q = 18$, the original PAM and TFP-PAM are used for 100 times respectively to cluster this database. The clustering accuracies and the iteration times are shown in Fig.8, Fig.9, Fig.10 and Fig.11.

We can see from the figures, the clustering accuracies of the two algorithms are similar, the average clustering accuracy of PAM is 73.29%, and the average clustering accuracy of TFP-PAM is 73.87%. As to the iteration times, the iteration times of PAM are fluctuant between 1 and 2, with an average value of 1.98, while the iteration times of TFP-PAM are fluctuant between 1 and 5, with an average value of 1.85. The average iteration times of TFP-PAM are similar, however, the time for each iteration in TFP-PAM is much lower than PAM.

Through these experiments, we can see TFP-PAM can obtain the results as good as the original PAM, while the time efficiency is highly improved.

Note that the accuracies of the cluster results are not very high, this is because the K-mediods method is suitable for the database with noises and outliers, while the two databases are normal databases. The two experiments are just used to compare the two algorithms.

7 Conclusions

An improved PAM algorithm, called TFP-PAM, is proposed for clustering. The algorithm uses a parallel mechanism in the TFMCPI-P systems. The time complexity of TFP-PAM is improved to $O(1)$ compared to other K-meodids algorithms. Experimental results, using the Iris database and the real short text database, show that TFP-PAM performs well in clustering. The results give

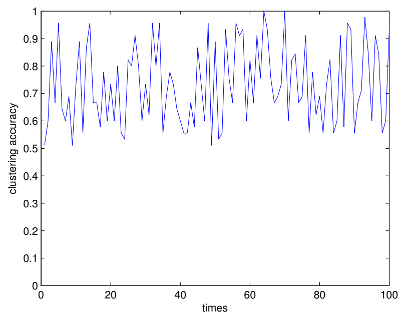


Fig. 8. The clustering accuracies of the original PAM

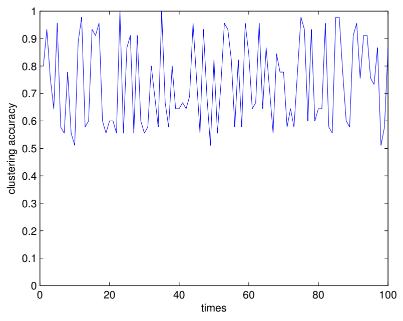


Fig. 9. The clustering accuracies of TFP-PAM

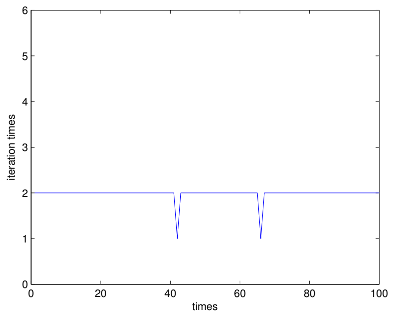


Fig. 10. The iteration times of the original PAM

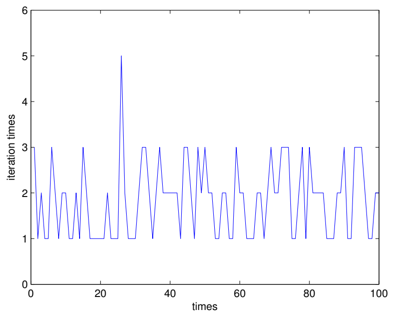


Fig. 11. The iteration times of TFP-PAM

some hints to improve conventional algorithms by using the parallel mechanism of membrane computing models.

For further research, more suitable databases should be found, and it is of interest to use some other interesting neural-like membrane computing models, such as the spiking neural P systems (SN P systems) [19], to improve the PAM algorithm. SN P systems are inspired by the mechanism of the neurons that communicate by transmitting spikes. The cells in SN P systems are neurons that have only one type of objects called spikes, which is easier to control in biological experiments. Zhang et al. [20, 21], Song et al. [22–33] and Zeng et al. [34] provided good examples. Also, some other data mining algorithms can be improved by using parallel evolution mechanisms and tree membrane structures, such as spectral clustering, support vector machines, and genetic algorithms [35].

References

1. Kaufman L, Rousseeuw P J. Finding Groups in Data: An Introduction to Cluster Analysis. Wiley, 2008.
2. Ng R T, Han J. Efficient and Effective Clustering Methods for Spatial Data Mining. International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc. 1994:144-155.
3. Păun, Gh. Computing with membranes. Journal of Computer and System Sciences, 2000, 61(1):108-143.
4. Păun, Gh., Rozenberg, G., Salomaa, A. The Oxford Handbook of Membrane Computing. Oxford University Press, 2010.
5. Pan, L., Păun, Gh., Song, B. Flat maximal parallelism in P systems with promoters. Theoretical Computer Science, 2016, 623:83-91.
6. Song, B., Pan, L., Pérez-Jiménez, M J. Tissue P systems with protein on cells. Fundamenta Informaticae, 2016, 144(1):77-107.
7. Zhang, X., Pan, L., Păun, A. On the universality of axon P systems. IEEE Transactions on Neural Networks and Learning Systems, 2015, 26(11):2816-2829.
8. Wang, J., Shi, P., Peng, H. Membrane computing model for IIR filter design. Information Sciences, 2016, 329: 164-176.
9. Singh, G., Deep, K. A new membrane algorithm using the rules of Particle Swarm Optimization incorporated within the framework of cell-like P-systems to solve Sudoku. Applied Soft Computing, 2016, 45:27-39.
10. Zhang, G., Rong, H., Cheng, J., Qin, Y. A population-membrane-system-inspired evolutionary algorithm for distribution network reconfiguration. Chinese Journal of Electronics, 2014, 23(3):437-441.
11. Peng, H., Wang, J., Pérez-Jiménez, M J, Riscos-Núñez, A. An unsupervised learning algorithm for membrane computing. Information Sciences, 2015, 304:80-91.
12. Zeng, X., Xu, L., Liu, X., Pan, L. On languages generated by spiking neural P systems with weights. Information Sciences, 2014, 278(10):423-433.
13. Liu, X., Li, Z., Liu, J., Liu, L., Zeng, X. Implementation of arithmetic operations with time-free spiking neural P systems. IEEE Transactions on Nanobioscience, 2015, 14(6):617-624.
14. Song, T., Zheng, P., Wong, M L D., Wang, X. Design of logic gates using spiking neural P systems with homogeneous neurons and astrocytes-like control. Information Sciences, 2016, 372:380-391.

15. Pan, L., Păun, G. On parallel array P systems automata, Universality, Computation. Springer International Publishing, 2015:171-181.
16. Song, T., Zheng, H., He, J. Solving Vertex Cover Problem by tissue P systems with cell division. Applied Mathematics and Information Science, 2014, 8(1):333-337.
17. Zhao, Y., Liu, X., Wang, W. ROCK clustering algorithm based on the P system with active membranes. Wseas Transactions on Computers, 2014, 13:289-299.
18. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.
19. Păun, Gh., Rozenberg, G., Salomaa, A. The Oxford Handbook of Membrane Computing. Oxford University Press, 2010.
20. Zhang, X., Wang, B., Pan, L. Spiking neural P systems with a generalized use of rules. Neural Computation, 2014, 26(12):2925-2943.
21. Zhang, X., Zeng, X., Luo, B., Pan, L. On some classes of sequential spiking neural P systems. Neural Computation, 2014, 26(5):974-997.
22. Song, T., Pan, L., Păun, Gh. Asynchronous spiking neural P systems with local synchronization. Information Sciences, 2012, 219:197-207.
23. Song T, Pan L. Spiking neural P systems with rules on synapses working in maximum spikes consumption strategy. IEEE Transactions on Nanobioscience. 2015 Jan;14(1):38–44.
24. Song T, Pan L. Spiking neural P systems with rules on synapses working in maximum spiking strategy. IEEE Transactions on Nanobioscience. 2015 Jun;14(4):465–477.
25. Song T, Zou Q, Zeng X, Liu X. Asynchronous spiking neural P systems with rules on synapses. Neurocomputing. 2015 Mar;151(1):1439–1445.
26. Song T, Xu J, Pan L. On the universality and non-nniversality of spiking neural P systems with rules on synapses. IEEE Transactions on Nanobioscience. 2015 Dec;14(8):960–966.
27. Song T, Zheng P, Wong M L D, Wang X. Design of logic gates using spiking neural P systems with homogeneous neurons and astrocytes-like control. Information Sciences. 2016 Dec; 372: 380-391.
28. Song T, Luo L, He J, Chen Z, Zhang K. Solving subset sum problems by time-free spiking neural P systems. Applied Mathematics and Information Sciences. 2014 Jan;8(1):327–332.
29. Song T, Zheng H, He J. Solving vertex cover problem by tissue P systems with cell division. Applied Mathematics and Information Science. 2014 Jan;8(1):333–337.
30. Song T, Pan L. Spiking neural P systems with request rules. Neurocomputing. 2016 Jun;193(12):193–200.
31. Wang X, Song T, Gong F, Zheng P. On the computational power of spiking neural P systems with self-organization. Scientific Reports. 2016 Jun;6:27624. doi: 10.1038/srep27624.
32. Song T, Wang X. Homogenous spiking neural P systems with inhibitory synapses. Neural Processing Letters. 2015 Aug;42(1):199–214.
33. Song T, Liu X, Zeng X. Asynchronous spiking neural P systems with anti-spikes. Neural Processing Letters. 2015 Dec;42(3):633–647.
34. Zeng, X., Zhang, X., Song, T., Pan, L. Spiking neural P systems with thresholds. Neural Computation, 2014, 26(7):1340-1361.
35. Han, J., Kambr, M., Pei, J. Data Mining: Concepts and Techniques. Elsevier Inc., 2012.

Deterministic Solutions to NP-Complete Problems using Numerical P Systems with Lower Thresholds

Ivan Cedric H. Macababayao, Erika Mika L. Amores, Nestine Hope S. Hernandez, and Francis George C. Cabarle

Algorithms & Complexity Lab
Department of Computer Science
University of the Philippines Diliman
Diliman 1101 Quezon City, Philippines
ihmacababayao, elamores, fccabarle@up.edu.ph,
nshernandez@dcs.upd.edu.ph

Abstract. Membrane computing is a branch of the field of natural computing that abstracts algorithms from the structures and functions of biological cells. Membrane systems, being parallel in nature, allows them to execute multiple sets of instructions or processes simultaneously. Numerical P Systems are a kind of hierarchical membrane system that use numerical values instead of objects in its compartments. With Numerical P Systems with thresholds being a fairly new variant of NP Systems, few solutions can be found that make use of it. Most of these being non-deterministic in nature - namely solutions to the `Subset Sum Problem` and the `Boolean Satisfiability Problem`. One way of making these deterministic in nature is with the use of a guide. By using a binary string generator, we are able to use the binary string as a guide to know which subsets to solve for or which variables will have a certain value. We show in this paper deterministic families of solutions to the `Subset Sum Problem` and the `Boolean Satisfiability Problem` that run in exponential time and polynomial time, using Numerical P Systems with Threshold.

1 Introduction

Membrane computing is a branch of natural computing wherein the structure and processes of living cells are abstracted for computing models [8][1]. These models, also known as P Systems, take advantage of the parallel nature of communication and computation between cells, which allows us to speed up the processing of information. This is a critical feature in terms of computation, as parallelism can have a significant positive impact in improving the time complexity of certain algorithms[9][10].

This particular feature of P Systems can be utilized in solving computationally hard problems. Finding solutions to computationally hard problems typically involve exhausting through every input possibilities until the right one (or none) is found. Through the parallelism of membrane systems, the amount of time needed to find a solution can be significantly reduced.

Numerical P Systems, first introduced in [7], are a kind of cell-like P Systems that uses concepts in economics, combined with the processes of living cells. One of its uses is in the design of robot controllers [2]. With this in mind, an extension of the Numerical P System is being used as a more powerful modeling tool for robot behaviors as it allows the existence of more than one active program or rule in each membrane while remaining deterministic in nature. This is the Enzymatic Numerical P System. It has enzyme-like variables allowing the existence of more than one program (rule) in each membrane, while keeping the deterministic nature of the system [13]. Another variant of Numerical P Systems is called Numerical P Systems with Threshold, first introduced in [14]. In this variant, every program in the system has a constant *threshold*, which will make the programs only fire if the values involved are higher (for lower threshold) or lower (for upper threshold) than that constant. This feature adds more control to the system in terms of deciding which programs will run every time step.

A deterministic algorithm is (potentially) implementable, or at least it is theoretically possible to implement or simulate with a deterministic machine. Thus, a deterministic approach is likely easier to simulate with a silicon-based computer. A disadvantage to this is that determinism usually increases the (time and/or space) complexity of the system.

Other P systems have been used to tackle with the `Subset Sum Problem` and the `Boolean Satisfiability Problem`. These include P Systems with Active Membranes and Spiking Neural P Systems. By using different extensions and playing around with the rules of certain systems, different nondeterministic and deterministic solutions have been presented in papers such as in [12][6][3] . In the case of Numerical P System with Thresholds (NPT, for short), there could still be a lot to discover as it is a fairly new P system. That being said, [5] is, so far, the only reference encountered that has presented solutions using the Numerical P System with Thresholds.

Our main reference for this paper is [5]. It contains nondeterministic solutions to the `Subset Sum Problem` and the `Boolean Satisfiability Problem` that act as the guide for the solutions that will be presented in the following sections. In this paper we use Numerical P Systems with Lower Threshold to provide deterministic families of solutions to the same two problems stated previously. Two sets of families will be provided: exponential time solutions and polynomial time solutions. This will be done by constructing Binary String Generators, which are small LTNP's that continuously output binary strings of a given length until all possible permutations are produced. These generators, when appended to non-deterministic verifiers to computationally hard problems, will be able to deterministically find solutions to said problems.

The following sections of this paper contain: the construct of numerical P systems with threshold and some preliminary definitions in Sections 2 and 3.1; non-deterministic families of solutions to `SSP` and `SAT` in Section 3.2; an introduction to the Binary String Generator can be seen in Section 4; the two sets of deterministic families of solutions to the `Subset Sum Problem` and the `Boolean Satisfiability Problem` are given in Sections 5 and 6, respectively; and lastly, we end with some final remarks and possible future research directions in Section 7.

2 Numerical P System with Threshold

A Numerical P System with Threshold Π is a tuple

$$(m, H, \mu, t, (Var_1, Pr_1, Var_1(0)), \ldots, (Var_m, Pr_m, Var_m(0)), Var_{in}, Var_{out}))$$

defined as follows:

- $m \geq 1$ is the number of membranes;
- H is an alphabet with m symbols;
- μ is the membrane structure with m membranes injectively labeled with the elements in H;
- t is a constant, called threshold;
- Var_i, $1 \leq i \leq m$ is the finite set of variables from region i;
- $Var_i(0)$, $1 \leq i \leq m$ is the set of initial values of the variables in region i;
- Pr_i, $1 \leq i \leq m$, is the finite set of programs in region i;
- Var_{in} and Var_{out} are the sets of input and output variables, respectively.

Each program $p_{j,i} \in Pr_i$ is of the form

$$F_{j,i}(x_{1,i}, \ldots, x_{k,i})|_T \rightarrow c_{j,1}|v_1 + \cdots + c_{j,n_i}|v_{n_i}$$

where $c_{j,1}, \ldots, c_{j,n_i}$ are natural numbers, denoting the jth program in region i with $\{x_{1,i}, \ldots, x_{k,i}\} \subseteq Var_i$ and $\{v_1, \ldots, v_{n_i}\} \subseteq Var_i \cup Var_{par(i)} \cup \bigcup_{ch \in Ch(i)} Var_{ch}$ where $par(i)$ is the parent membrane of membrane i and $Ch(i)$ is the set of child membranes of membrane i [14].

A program has two parts: the production function and the repartition protocol. The left side ($F_{j,i}(x_{1,i}, \ldots, x_{k,i})$) comprises the production function which consists of the rules or the variables to be used. It is considered active if the values of the variables are either greater than or equal to the threshold (if it is a lower threshold) or less than or equal to the threshold (if it is an upper threshold). The right side ($c_{j,1}|v_1 + \cdots + c_{j,n_i}|v_{n_i}$) comprises the repartition protocol where for each variable v_{n_i}, a coefficient c_{j,n_i} is associated to it. This distributes the unitary portion (the quotient of the computed value of the production function and the sum of all the coefficients in the repartition protocol) computed to each of the variables present in the repartition protocol according to the assigned coefficients of these programs.

Adapted from [13] and [7], there are three modes in how a repartition protocol of a program behaves. The first is called *seq (sequential)*, where only one active program is nondeterministically chosen to execute every region. The second is called *oneP (one-parallel)*, where a maximal set of active programs is nondeterministically chosen to execute every region, provided that these chosen programs do not share a common variable. The third, called *allP (all-parallel)*, executes all active programs, regardless if a common variable is shared. *allP* mode indicates a deterministic approach in executing the programs.

All deterministic Numerical P Systems with Threshold used in this paper follow the *allP* mode. The non-deterministic Numerical P Systems in Section 3.2 follow the *oneP* mode.

3 NPT systems

3.1 Recognizer NPT systems

To determine whether a P system is able to solve a given problem, a recognizer P system is used. The following definition of a recognizer NPT system is modified to cater to numerical values instead of a multiset of objects [5] [11].

Definition 1. *A recognizer NPT system Π is a P system such that: (a) the set of output variables contains a distinguished variable* `output`*; (b) all computations halt; and (c) if C is a computation of Π, then the value of* `output` *is either a 1 or a 0 to signal acceptance or rejection, respectively, at the last step of the computation. 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.*

To determine whether a P system is able to gain the desired result, a family of P systems is also used. A decision problem can be represented as a pair $Y = (I_Y, \theta_Y)$ where I_Y is a language over a finite alphabet and θ_Y is a total boolean function over I_Y. An instance of a decision problem can be represented by a pair (cod, s) where $s \in \mathbb{N}$ and cod refers to an encoding of the instance which will be placed as initial values of the input variables in the initial configuration [5].

A family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{Z}^+\}$, of P systems (specifically NPT systems in our context) is a set of P systems that takes a parameter n to construct each system.

Definition 2. *A family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{Z}^+\}$, of NPT systems, solves a problem (I_Y, θ_Y) if there exists a pair (cod, s) over I_Y such that for each instance $u \in I_Y$:*

1. *$n = s(u) \in \mathbb{N}$ and $cod(u)$ gives the initial values for the input variables of $\Pi(n)$;*
2. *there exists an accepting computation of $\Pi(n)$ with input $cod(u)$ if and only if $\theta_Y(u) = 1$.*

3.2 Non-deterministic NPT for SSP and SAT

In this section we show non-deterministic Numerical P System with Lower Threshold solutions for SSP and SAT, first introduced in [5]. These systems will later on be used as basis for other families of solutions.

The construct of a LTNP system solving an instance $I = ((\dot{x_1}, \dot{x_2}, \cdots, \dot{x_n}), \dot{t})$ of **SSP** of size n is as follows:

$\Pi_{\mathsf{SSP}}(n) = (1, \{1\}, [_1]_1, 1, (Var_1, Pr_1, Var_1(0)), Var_{in}, Var_{out})$

– $Var_1 = \{x_1, x_2, \cdots, x_n, t, v_1, v_2, \cdots, v_n, g, b_1, b_2, \cdots, b_n,$
$t_d, t_D, s_d, s_D, d, D, \mathsf{output}\}$;
– Pr_1 is composed of programs of the form $p_{1,1}, p_{2,1}, p_{3,1}, p_{4,1}, p_{5,1}, p_{6,1}, p_{7,1}$ and $p_{8,1}$ where:
- $p_{1,1} : x_i|_1 \to 1|v_i, 1 \le i \le n,$
- $p_{2,1} : x_i|_1 \to 1|g, 1 \le i \le n,$
- $p_{3,1} : b_i|_1 \to 1|v_i, 1 \le i \le n,$
- $p_{4,1} : 2t|_1 \to 1|t_d + 1|t_D,$
- $p_{5,1} : 2\sum_{i=1}^{n}(v_i - 1)|_1 \to 1|s_d + 1|s_D,$
- $p_{6,1} : t_d - s_d + 1|_1 \to 1|d,$
- $p_{7,1} : s_D - t_D + 1|_1 \to 1|D,$
- $p_{8,1} : d - D + 1|_1 \to 1|\mathsf{output};$

– $Var_1(0) = (\dot{x_1}, \dot{x_2}, \cdots, \dot{x_n}, \dot{t}, 0, \overset{n}{\cdots}, 0, 0, 1, \overset{n}{\cdots}, 1, 0, 0, 0, 0, 0, 0, 0)$;
– $Var_{in} = \{x_1, x_2, \cdots, x_n, t\}$;
– $Var_{out} = \{\mathsf{output}\}$.

The construct of the LTNP system solving an instance $I = \phi$ of **SAT** having n boolean variables $(x_1, x_2, \cdots, x_n)$ and k clauses is as follows:

$\Pi_I = (1, \{1\}, [_1]_1, 1, (Var_1, Pr_1, Var_1(0)), Var_{in}, Var_{out})$
where

– $Var_1 = \{T_j, 1 \le j \le n\} \cup \{v_j, 1 \le j \le n\} \cup \{w_j, 1 \le j \le n\} \cup$
$\{x_{ji}, 1 \le j \le n, 1 \le i \le k\} \cup \{\bar{x}_{ji}, 1 \le j \le n, 1 \le i \le k\} \cup$
$\{c_i, 1 \le i \le k\} \cup \{sum, count, \mathsf{output}\}$;
– Pr_1 is composed of programs of the form $p_{1,1}, p_{2,1}, p_{3,1}, p_{4,1}, p_{5,1}, p_{6,1}$ and $p_{7,1}$ where
- $p_{1,1} : 2T_j|_1 \to 1|v_j + 1|w_j, 1 \le j \le n,$
- $p_{2,1} : 2(T_j - 1)|_1 \to 1|v_j + 1|w_j, 1 \le j \le n,$
- $p_{3,1} : k\ v_j|_1 \to \sum_{i=1}^{k} 1|x_{ji}, 1 \le j \le n,$
- $p_{4,1} : k(-w_j + 3)|_1 \to \sum_{i=1}^{k} 1|\bar{x}_{ji}, 1 \le j \le n,$

- $p_{5,1}$: for each clause we include a program of the following form

$$\sum x_{ji} + \sum \bar{x}_{ji}|_1 \rightarrow 1|c_i$$

where x_{ji} (resp, $\bar{x}_{ji}$) is included in the sum if and only if the literal x_j (resp, $\bar{x}_j$) is in clause i;

- $p_{6,1}$: $2\sum_{i=1}^{k} c_i|_1 \rightarrow 1|sum + 1|count$,
- $p_{7,1}$: $sum - count + 1|_1 \rightarrow 1|\mathsf{output}$;

– $Var_1(0) = (2, \overset{n}{\cdots}, 2, 0, \overset{n}{\cdots}, 0, 0, \overset{n}{\cdots}, 0, 0, \overset{nk}{\cdots}, 0, 0, \overset{nk}{\cdots}, 0,$
$-|S_1|, -|S_2|, \cdots, -|S_k|, 0, 0, 0)$
where $|S_i|$ is the number of literals in clause i;
– $Var_{in} = \{c_i, 1 \leq i \leq k\}$;
– $Var_{out} = \{\mathsf{output}\}$.

The above systems rely on a non-deterministic guess of input values. In order to remove this non-determinism, we need to be able to produce all possible configurations of input, and verify each one of them.

In the next chapter, we introduce a way to do this.

4 Binary String Generator

The main idea of a Binary String Generator (BSG) is to convert nondeterministic solutions to deterministic ones. With the binary string as a guide, it can be used to determine which of the elements in a set will be part of a subset or whether a certain element has the value of $TRUE$ or $FALSE$, corresponding to bit 1 or 0, respectively. Thus, the Binary String Generators produce all possible permutations of 1's and 0's, given the length of the string.

The BSG's that will be introduced in this paper are all deterministic LTNP Systems with a constant threshold 1. By default, all Binary String Generators used in this paper are in *allP*.

The output of a BSG is represented by a list of variables that contain 1's and 0's. These variables are the x_n's (for Exponential Time Binary String Generators) and the $X_{p,n}$'s (for Linear Time Binary String Generators). These outputs will be used as input to a verifier, which will use the generated binary strings as basis to solve a certain (computationally hard) problem. Note that the BSG's have no formal output, i.e., they return nothing when they halt. The verifiers are expected to receive these outputs every turn.

These will be used to provide deterministic Numerical P System with Threshold families of solutions to the `Subset Sum Problem` and the `Boolean Satisfiability Problem`.

In this paper, we will be presenting two different kinds of Binary String Generators.

4.1 Exponential Time Binary String Generator

Here we introduce a generator that produces one binary string for each time step taken. The generated binary string will be fed directly to the verifier.

All possible binary strings of length n are produced in 2^n number of steps. We call this the Exponential Time Binary String Generator (EBSG).

The EBSG is a deterministic LTNP with a constant threshold of 1, which accepts no input. The output is defined implicitly as the string $\{x_n, x_{n-1}, \cdots, x_1\}$. This string changes every time step until it reaches $\{1, 1, \cdots, 1\}$, where the system will halt. No output will be returned when EBSG halts.
The construct of an EBSG that produces strings of size n is as follows:

$$\Pi^D_{\mathtt{EBSG}}(n) = (1, \{1\}, [_1]_1, 1, (Var_1, Pr_1, Var_1(0)), Var_{in}, Var_{out})$$

where

- $Var_1 = \{x_1, x_2, \cdots, x_n, u_1, u_2, \cdots, u_n\}$;
- Pr_1 is composed of programs of the following forms:
  - for each i we include a program of the following form

    $$p_1 : i(u_i)(x_{i-1})(x_{i-2}) \cdots (x_1)|_1 \to 1|x_i + 1|u_{i-1} + 1|u_{i-2} + \cdots + 1|u_1$$

    where $1 \le i \le n$;
  - for all i, j we include a program of the following form

    $$p_2 : x_j(u_i)(x_{i-1})(x_{i-2})...(x_1)|_1 \to 1|x_j$$

    where $j > i$ and $1 \le i, j \le n$;
- $Var_1(0) = \{0, 0, \overset{n}{\cdots}, 0, 1, 1, \overset{n}{\cdots}, 1\}$;
- $Var_{in} = \{\}$;
- $Var_{out} = \{\}$.

The programs of the first type generate the binary string while programs of the second type replenish certain binary strings after it is used. This is done so as to not affect the outcome of the next time steps.

EBSG has $2n$ variables, $n + \dfrac{n(n-1)}{2}$ programs, and runs in at most $2^n + 1$ steps where n is the length of input.

Theorem 1. *There exists deterministic binary string generators having 1 membrane, $2n$ variables, $n + \dfrac{n(n-1)}{2}$ programs with polynomial production functions and working in allP application mode, that runs in at most 2^n steps where n is the size of the input set S.*

step	$x_4x_3x_2x_1$
0	0000
1	0001
2	0010
3	0011
...	...
15	1111

Table 1. Output table for EBSG, $n = 4$.

4.2 Linear Time Binary String Generator

We introduce a second kind of Binary String Generator, called Linear Time Binary String Generator (LBSG). In this generator the number of output strings get doubled every time step, that is, one output for the first iteration, two outputs for the second, four outputs for the third, and so on. Thus, the number of strings generated per iteration is 2^{t-1}, where t is the current time step. All possible binary strings of length n are produced in n number of steps, with an additional step that deals with halting.
The construct of a LBSG of size n is as follows:

$$\Pi^{D}_{\texttt{LBSG}}(n) = (1, \{1\}, [_1]_1, 1, (Var_1, Pr_1, Var_1(0)), Var_{in}, Var_{out})$$

where

- $Var_1 = \{c, v_1, v_2, \cdots, v_n, \overline{v_1}, \overline{v_2}, \cdots, \overline{v_n}, X_{1,1}, X_{1,2}, \cdots, X_{2,1}, X_{2,2}, \cdots, X_{2^{n-1},n}, h\}$;
- Pr_1 is composed of programs of the following forms:
  - p_1: $c + 1|_1 \rightarrow 1|c$
  - for all i, where $1 \leq i \leq n$, we add the following rules:
    - p_2: $(c - i + 1)(i - c + 1)|_1 \rightarrow 1|v_i$
    - p_3: $-(c - i + 1)(i - c + 1)|_1 \rightarrow 1|\overline{v_i}$
    - p_4: $\overline{v_i}|_1 \rightarrow 1|v_i$
  - for all i, p, where $1 \leq i < n$ and $1 \leq p \leq 2^{n-1}$, we add the following rule:

    p_5: $3(v_i)(X_{p,i})|_1 \rightarrow 1|X_{2p,i} + 1|X_{2p,i+1} + 1|X_{2p-1,i+1}$

  - for all i, j, p, where $1 \leq i < n$, $1 \leq p \leq 2^{n-1}$ and $1 \leq j < i$, we add the following rule:

    p_6: $2(v_i)(X_{p,i})(X_{p,j})|_1 \rightarrow 1|X_{2p-1,j} + 1|X_{2p,j}$

  - for all p where $1 \leq p \leq 2^{n-1}$, we add the following rule:

    $p_{7,1}$: $(X_{p,1})(X_{p,2}) \cdots (X_{p,n})|_1 \rightarrow 1|h$

  - $p_{8,1}$: $-(c + h)|_1 \rightarrow 1|c$;

– $Var_1(0) = (1, 0, \overset{n}{\cdots}, 0, 0, \overset{n}{\cdots}, 0, 0, \overset{(2^{n-1})(n)}{\cdots}, 1, 0)$;
– $Var_{in} = \{\}$;
– $Var_{out} = \{\}$.

LBSG works as follows: p_1 fires every time step, incrementing c every turn. This c represents the number of time steps, plus one. p_2 and p_3 also fire every time step. Note that $v_i = 1$ if and only if $i = c + 1$, and that $v_i < 0$ otherwise. Also note that the value added to $\overline{v_i}$ is always the negative of v_i. Rule p_4 uses the values of $\overline{v_i}$ to reset the values of v_i to 0. Rules p_5 and p_6 are responsible for the production of the next output (to be used in the next time step). Each current output string is used to produce two more output strings for the next step (for brevity we call these new strings P and $2P$). In both strings the bit $X_{p,c}$ (which is always 1) is copied to $X_{p,c+1}$ (and $X_{2p,c+1}$). For P, $X_{p,c}$ becomes a 0, while it becomes a 1 for $2P$. The rest of the original string is copied without change.

A visual representation of the output per time step is as follows:

c	$X_{43}X_{42}X_{41}$	$X_{33}X_{32}X_{31}$	$X_{23}X_{22}X_{21}$	$X_{13}X_{12}X_{11}$
1				001
2			010	011
3	100	110	101	111

Table 2. Output table for LBSG, $n = 3$.

LBSG begins with 2^0 outputs. In the next step, we produce 2^1 outputs, then 2^2 outputs, then 2^3 and so on. In the $(n-1)^{th}$ step, we produce 2^{n-1} outputs. By this time, all in all we have already produced $2^0 + 2^1 + 2^2 + \cdots + 2^{n-1} = \dfrac{1-2^n}{1-2}$ output strings in $n - 1$ number of steps. This is equal to $2^n - 1$ output strings in $n - 1$ steps. Taking account of the first step, where the values of v_i is set up, and the last step, where we control the halting, we get $n + 1$ steps. Therefore, LBSG is able to produce all binary strings of length n in $n + 1$ number of steps.

Theorem 2. *There exists deterministic binary string generators having 1 membrane, $2^{n-1}n + 2n + 2$ variables, $2^{(n-1)}[\dfrac{n(n-1)}{2} + n] + 3n + 3$ programs with polynomial production functions and working in allP application mode, that runs in at most $n + 1$ steps where n is the size of the input set S.*

Nondeterministic families of solutions for two NP-Complete problems have been introduced in [5], also shown in Section 3.2. These problems are the `Subset Sum Problem` (SSP) and the `Boolean Satisfiability Problem` (SAT). A modified version of the said families of solutions will be used as verifiers, and will be appended to the Binary String Generators to provide deterministic families of solutions to the two problems.

5 Solutions to `Subset Sum Problem`

We first give the definition of the `Subset Sum Problem` **(SSP)** taken from [4]

Definition 3. `Subset Sum Problem` **(SSP)** *Given a (multi)set of positive integers $S = \{x_1, x_2, \ldots, x_n\}$ and a positive integer t. Is there a sub(multi)set $V \subseteq S$ such that $\sum\limits_{v \in V} v = t$?*

5.1 A solution to `Subset Sum Problem` using LTNP in exponential time

The construct of a deterministic LTNP system solving an instance $I = ((\dot{r_1}, \dot{r_2}, \cdots, \dot{r_n}), \dot{t})$ of **SSP** of size n is as follows:

$\Pi^D_{\mathsf{SSP}}(n) = (1, \{1\}, [_1]_1, 1, (Var_1, Pr_1, Var_1(0)), Var_{in}, Var_{out})$
where

– $Var_1 = \{r_1, r_2, \cdots, r_n, t, x_1, x_2, \cdots, x_n, h, u_1, u_2, \cdots, u_n, s, d, D, y, \mathsf{output}\}$;
– Pr_1 is composed of programs of the following forms:
- $p_{1,1} : r_i|_1 \to 1|r_i, 1 \le i \le n$,
- $p_{2,1} : t|_1 \to 1|t$,
- $p_{3,1} : h(-t)|_1 \to 1|t$,
- $p_{4,1} : h(-r_i)|_1 \to 1|r_i, 1 \le i \le n$,
- $p_{5,1}$: for each i we include a program of the following form

$$i(u_i)(x_{i-1})(x_{i-2}) \cdots (x_1)|_1 \to 1|x_i + 1|u_{i-1} + 1|u_{i-2} + \cdots + 1|u_1$$

  where $1 \le i \le n$;
- $p_{6,1}$: for all i, j we include a program of the following form

$$x_j(u_i)(x_{i-1})(x_{i-2})...(x_1)|_1 \to 1|x_j$$

  where $j > i$ and $1 \le i, j \le n$;
- $p_{7,1} : x_i(r_i)|_1 \to 1|s, 1 \le i \le n$,
- $p_{8,1} : t - s + 1|_1 \to 1|d$,
- $p_{9,1} : s - t + 1|_1 \to 1|D$,
- $p_{10,1} : d - 2|_1 \to 1|D$,
- $p_{11,1} : D - 2|_1 \to 1|d$,
- $p_{12,1} : d - D + 1|_1 \to 1|y$,
- $p_{13,1} : y|_1 \to 1|\mathsf{output}$,
- $p_{14,1} : y|_1 \to 1|h$,
- $p_{15,1} : x_1 \cdots x_n|_1 \to 1|h$;

– $Var_1(0) = (\dot{r_1}, \dot{r_2}, \cdots, \dot{r_n}, \dot{t}, 0, \overset{n}{\cdots}, 0, 0, 1, \overset{n}{\cdots}, 1, 0, 0, 0, 0, 0)$;
– $Var_{in} = \{r_1, r_2, \cdots, r_n, t\}$;
– $Var_{out} = \{\mathsf{output}\}$.

The encoding of instance I is placed as the initial values of the $n+1$ input variables. The remaining variables will all have an initial value of 0 except for each $u_i, 1 \leq i \leq n$ which has an initial value of 1.

Computation proceeds as follows: The first step involves generating the first string combination of 1's and 0's represented by the values of $x_1...x_n$. This will be used as the guide as to which combination of the input variables $r_1...r_n$ will be used for computation in the next time step. The values of $r_1...r_n$ and t will also be replenished so as to have the same value whether it will be used or not. This is to ensure that the input variables stay constant all throughout. Note that only programs $p_{1,1}, p_{2,1}$, and $p_{5,1}$ are applicable at this step. But for each time step, only one of the programs of $p_{5,1}$ will be applicable. These programs will be applicable on each time step unless the value of h becomes 1.

At the second step, in program $p_{7,1}$, the values of $x_1...x_n$ will be multiplied to the respective values of $r_1...r_n$. The products would then be added and stored to s. This is the sum of the values of the certain subset combination generated at the previous step. At the same time, a new string of 1's and 0's is being generated.

At the third step, we compute for the difference between the desired total t and the sum obtained at the previous step stored in s. Note that value $t-s$ is just the negative of $s-t$ such that they compute for pairs of numbers that are in $\{\{0,0\},\{-1,1\},\{-2,2\},\{-3,3\},\ldots\}$. With a constant 1 added to these values in programs $p_{8,1}$ and $p_{9,1}$, d and D will have values that are in $\{\{1,1\},\{0,2\},\{-1,3\},\{-2,4\},\ldots\}$. Note that one program of $p_{6,1}$ is also applicable in this step. It will replenish the used up value of x_i so as to not destroy the results of program $p_{5,1}$.

At the fourth step, $p_{12,1}$ will only be active if and only if both d and D have values greater than or equal to threshold 1. This happens only when the difference between the desired total value and the actual sum obtained from the deterministic choice of a subset V of the input S is 0. Programs $p_{10,1}$ and $p_{11,1}$ will also be applicable depending on the values of d and D. These steps ensure that the values of d and D, should one of them be a negative value, are reset to 0 before adding the new values.

Finally, should y contain a value of 1, this will be added to the output variable output. This means that a subset was obtained such that the given total and the computed sum is equal. The value of y will also be added to h which will trigger the halting of the constant replenishing of the input variables at the next time step. Otherwise, the computation will continue on from the second step until either a certain subset which will satisfy the given total is found or the values of $x_1...x_n$ are all 1's. This will render the program $p_{15,1}$ applicable which will add a value of 1 to h. This will also trigger the halting of the constant replenishing of the input variables at the next time step.

Corollary 1. *There exists a family of deterministic LTNP systems solving* **SSP** *having 1 membrane,* $3n+7$ *variables,* $4n+10+\dfrac{n(n-1)}{2}$ *programs with polynomial* *production functions and working in allP application mode, that runs in at most $2^n + 6$ steps where n is the size of the input set S.*

5.2 A solution to `Subset Sum Problem` using LTNP in polynomial time

The construct of a deterministic LTNP system solving an instance $I = ((\dot{r_1}, \dot{r_2}, \cdots, \dot{r_n}), \dot{t})$ of **SSP** of size n is as follows:

$$\Pi^{D}_{\mathsf{SSP}}(n) = (1, \{1\}, [_1]_1, 1, (Var_1, Pr_1, Var_1(0)), Var_{in}, Var_{out})$$

where

- $Var_1 = \{c, r_1, r_2, \cdots, r_n, t, X_{1,1}, X_{1,2}, \cdots, X_{2^{n-1},n}, v_1, v_2, \cdots, v_n,$
  $\overline{v_1}, \overline{v_2}, \cdots, \overline{v_n}, s_1, s_2, \cdots, s_{2^{n-1}}, d_1, d_2, \cdots, d_{2^{n-1}},$
  $D_1, D_2, \cdots, D_{2^{n-1}}, h, y, \mathsf{output}\}$;
- Pr_1 is composed of programs of the following forms:
  - $p_{1,1} : c + 1|_1 \rightarrow 1|c,$
  - $p_{2,1} : (c - i + 1)(i - c + 1)|_1 \rightarrow 1|v_i, 1 \le i \le n,$
  - $p_{3,1} : -(c - i + 1)(i - c + 1)|_1 \rightarrow 1|\bar{v_i}, 1 \le i \le n,$
  - $p_{4,1} : \overline{v_i}|_1 \rightarrow 1|v_i, 1 \le i \le n,$

  - $p_{5,1} : 3(v_i)(X_{p,i})|_1 \rightarrow 1|X_{2p,i} + 1|X_{2p,i+1} + 1|X_{2p-1,i+1},$
    $1 \le i \le n$ and $1 \le p \le 2^{n-1},$
  - $p_{6,1} : 2(v_i)(X_{p,i})(X_{p,j})|_1 \rightarrow 1|X_{2p-1,j} + 1|X_{2p,j},$
    $j < i, 1 \le i \le n,$ and $1 \le p \le 2^{n-1},$

  - $p_{7,1} : (X_{p,1})(X_{p,2}) \cdots (X_{p,n})|_1 \rightarrow 1|h, 1 \le p \le 2^{n-1},$
  - $p_{8,1} : -(c + h)|_1 \rightarrow 1|c,$

  - $p_{9,1} : (X_{p,i})(r_i)|_1 \rightarrow 1|s_p, 1 \le i \le n$ and $1 \le p \le 2^{n-1},$
  - $p_{10,1} : t - s_p + 1|_1 \rightarrow 1|d_p, 1 \le p \le 2^{n-1},$
  - $p_{11,1} : s_p - t + 1|_1 \rightarrow 1|D_p, 1 \le p \le 2^{n-1},$
  - $p_{12,1} : d_p - 2|_1 \rightarrow 1|D_p, 1 \le p \le 2^{n-1},$
  - $p_{13,1} : D_p - 2|_1 \rightarrow 1|d_p, 1 \le p \le 2^{n-1},$
  - $p_{14,1} : 2(d_p - D_p + 1)|_1 \rightarrow 1|\mathsf{output} + 1|h, 1 \le p \le 2^{n-1},$

  - $p_{15,1} : t|_1 \rightarrow 1|t,$
  - $p_{16,1} : r_i|_1 \rightarrow 1|r_i, 1 \le i \le n,$
  - $p_{17,1} : h(-t)|_1 \rightarrow 1|t,$
  - $p_{18,1} : h(-r_i)|_1 \rightarrow 1|r_i, 1 \le i \le n;$
- $Var_1(0) = (1, \dot{r_1}, \dot{r_2}, \cdots, \dot{r_n}, \dot{t}, 0, \overset{(n)(2^{n-1})}{\cdots}, 1, 0, \overset{n}{\cdots}, 0, 0, \overset{n}{\cdots}, 0, 0, \overset{2^{n-1}}{\cdots}, 0,$
  $0, \overset{2^{n-1}}{\cdots}, 0, 0, \overset{2^{n-1}}{\cdots}, 0, 0, 0);$
- $Var_{in} = \{r_1, r_2, \cdots, r_n, t\};$
- $Var_{out} = \{\mathsf{output}\}.$

The variables c and $X_{1,1}$ are equal to 1, while the variables r_1, r_2, up to r_n, as well as t come from the input. The rest of the variables are all equal to 0.

The following is the sequence of processes:

In the first step, programs $p_{1,1}$ to $p_{3,1}$ fire. This goes on until the whole process is halted. $p_{1,1}$ simply increments to c. $p_{2,1}$ puts a value in v_i such that $v_i = 1$ when $c = i$ and $v_i \leq 0$ otherwise. $p_{3,1}$ adds the negative of v_i to $\overline{v_i}$. Also in the first step, $p_{9,1}$ fires. In this rule the product of $X_{p,i}$ and r_i is added to s_p (Initially, $(X_{1,i})(r_i)|_1 \rightarrow s_1$ fires). This is the summation of all subsets flagged as 1 in the current permutation.

In the next step, programs $p_{4,1}$ and $p_{5,1}$ (and in other iterations $p_{6,1}$) fire. $p_{10,1}$ and $p_{11,1}$ also fire. $p_{4,1}$ adds the value of $\overline{v_i}$ to v_i. Notice that $v_i \leq 0$ for all $i \neq b$ after this rule. Rules $p_{5,1}$ and $p_{6,1}$ are responsible for the evolution and multiplication of input. Refer to Section 4.2 for the details of this. $p_{10,1}$ and $p_{11,1}$ adds to d_p and D_p such that d_p and D_p will be equal to 1 if and only if $s_p = t$. Otherwise, one of them will have a negative value.

Then, either $p_{12,1}$ or $p_{13,1}$ fires. $p_{12,1}$ and $p_{13,1}$ resets the value d_p and D_p for the next step. $p_{14,1}$ will also fire if and only if d_p and D_p are greater than or equal to 1. Note that this rule gives a value of 1 to *output* and h.

$p_{15,1}$ and $p_{16,1}$ are rules that fire every time step until halted. These rules replenish the values of t and all r_i, ensuring that the next permutations get the correct value.

Rule $p_{7,1}$ will fire when the string $11 \cdots 1$ is encountered, which means that all permutations has been accounted for. This will add a value of 1 to h, and signifies that the process will halt in the next step.

$p_{8,1}$, $p_{17,1}$ and $p_{18,1}$ only fire when $h = 1$. These make the values of c, t, and all r_i less than 1. These set of rules ensure that no "infinite loop" occurs.

Note that the above set of steps will be performed every time step (if applicable) from the moment they were first triggered, until halted. For example, the first step will be repeated every time step, along with the other steps.

Corollary 2. *There exists a family of deterministic LTNP systems solving* **SSP** *having 1 membrane,* $(n+3)(2^{n-1}) + 3n + 5$ *variables,* $(2^{n-1})[\frac{n(n-1)}{2} + 2n + 6] + 5n + 4$ *programs with polynomial production functions and working in allP application mode, that runs in at most* $n+6$ *steps where* n *is the size of the input set* S*.*

6 Solutions to `Boolean Satisfiability Problem`

We now give the definition of the `Boolean Satisfiability Problem` **(SAT)** [4].

Definition 4. `Boolean Satisfiability Problem` **(SAT)** *Given a boolean formula* ϕ *in conjunctive normal form having* n *boolean variables and* k *clauses. Is* ϕ *satisfiable? That is, can we assign truth values to the* n *boolean variables such that* ϕ *evaluates to true?*

For the next set of solutions, the structure of the boolean formula will be included in the input. The formula must then be represented in a form applicable to LTNP.

If $Var_{j,i} = 1$, then X_i is in clause j. Similarly, if $\overline{Var_{j,i}} = 1$, then $\overline{X_i}$ is in clause j. Thus, $(X_1 \vee X_3 \vee \overline{X_4}) \wedge (X_2 \vee X_4) \wedge (\overline{X_1} \vee X_2 \vee X_3 \vee X_4)$ can be represented as:

$Var_{j,i}$	$i = 1$	2	3	4
$j = 1$	1	0	1	0
2	0	1	0	1
3	0	1	1	1

Table 3. Table representation for $Var_{j,i}$.

$\overline{Var_{j,i}}$	$i = 1$	2	3	4
$j = 1$	0	0	0	1
2	0	0	0	0
3	1	0	0	0

Table 4. Table representation for $\overline{Var_{j,i}}$.

6.1 A solution to `Boolean Satisfiability Problem` using LTNP in exponential time

The construct of the deterministic LTNP system solving an instance $I = \phi$ of **SAT** having n boolean variables and k clauses is as follows:

$\Pi_I^D = (1, \{1\}, [_1]_1, 1, (Var_1, Pr_1, Var_1(0)), Var_{in}, Var_{out})$
where

– $Var_1 = \{T_i, 1 \leq i \leq n\} \cup \{t_i, 1 \leq i \leq n\} \cup \{u_i, 1 \leq i \leq n\} \cup$
$\{k_{j,i}, 1 \leq i \leq n, 1 \leq j \leq k\} \cup \{\overline{k}_{j,i}, 1 \leq i \leq n, 1 \leq j \leq k\} \cup$
$\{C_j, 1 \leq j \leq k\} \cup \{min_j, 1 \leq j \leq k\} \cup \{V_j, 1 \leq j \leq k\} \cup$
$\{Var_{j,i}, 1 \leq i \leq n, 1 \leq j \leq k\} \cup \{\overline{Var_{j,i}}, 1 \leq i \leq n, 1 \leq j \leq k\} \cup$
$\{g, h, synch, \mathsf{output}\}$;

– Pr_1 is composed of programs of the following forms:

- $p_{1,1}$: for each i we include a program of the following form

$$i(u_i)(t_{i-1})...(t_1)|_1 \rightarrow 1|t_i + 1|u_{i-1} + ... + 1|u_1$$

where $1 \leq i \leq n$;

- $p_{2,1}$: for all i we include a program of the following form

$$t_j(u_i)(t_{i-1})...(t_1)|_1 \rightarrow 1|t_i$$

  where $j > i$ and $1 \leq i \leq n$;
- $p_{3,1} : t_1...t_n|_1 \rightarrow 1|h$,
- $p_{4,1} : t_i|_1 \rightarrow 1|T_i, 1 \leq i \leq n$,
- $p_{5,1} : (n+1)synch|_1 \rightarrow 1|T_1 + ... + 1|T_n + 1|synch$,
- $p_{6,1} :$ $(k_{j,i})(T_i)|_1 \rightarrow 1|C_j, 1 \leq i \leq n$ and $1 \leq j \leq k$,
- $p_{7,1} : (\overline{k_{j,i}})(-T_i + 3)|_1 \rightarrow 1|C_j, 1 \leq i \leq n$ and $1 \leq j \leq k$,
- $p_{8,1} : C_j - min_j|_1 \rightarrow 1|V_j, 1 \leq j \leq k$,
- $p_{9,1} : 2(V_k)(V_{k-1}) \cdots (V_1)|_1 \rightarrow 1|\mathsf{output} + 1|h$,
- $p_{10,1} : V_j|_1 \rightarrow 1|g, 1 \leq j \leq k$,
- $p_{11,1} : 2Var_{j,i}|_1 \rightarrow 1|k_{j,i} + 1|min_j, 1 \leq i \leq n$ and $1 \leq j \leq k$,
- $p_{12,1} : 2\overline{Var_{j,i}}|_1 \rightarrow 1|\overline{k_{j,i}} + 1|min_j, 1 \leq i \leq n$ and $1 \leq j \leq k$,
- $p_{13,1} : k_{j,i}|_1 \rightarrow 1|k_{j,i}, 1 \leq i \leq n$ and $1 \leq j \leq k$,
- $p_{14,1} : \overline{k_{j,i}}|_1 \rightarrow 1|\overline{k_{j,i}}, 1 \leq i \leq n$ and $1 \leq j \leq k$,
- $p_{15,1} : min_j|_1 \rightarrow 1|min_j, 1 \leq j \leq k$,
- $p_{16,1} : h(-k_{j,i})|_1 \rightarrow 1|k_{j,i}, 1 \leq i \leq n$ and $1 \leq j \leq k$,
- $p_{17,1} : h(-\overline{k_{j,i}})|_1 \rightarrow 1|\overline{k_{j,i}}, 1 \leq i \leq n$ and $1 \leq j \leq k$,
- $p_{18,1} : h(-min_j)|_1 \rightarrow 1|min_j, 1 \leq j \leq k$,
- $p_{19,1} : h(-synch)|_1 \rightarrow 1|synch$;

– $Var_1(0) = (0, \overset{n}{\cdots}, 0, 0, \overset{n}{\cdots}, 0, 1, \overset{n}{\cdots}, 1, 0, \overset{nk}{\cdots}, 0, 0, \overset{nk}{\cdots}, 0, 0, \overset{k}{\cdots}, 0, 0, \overset{k}{\cdots}, 0,$
$0, \overset{k}{\cdots}, 0, Var_{1,1}, \overset{nk}{\cdots}, Var_{j,i}, \overline{Var_{1,1}}, \overset{nk}{\cdots}, \overline{Var_{j,i}}, 0, 0, 1, 0)$;
– $Var_{in} = \{Var_{j,i}, \overline{Var_{j,i}}, 1 \leq j \leq k, 1 \leq i \leq n\}$;
– $Var_{out} = \{\mathsf{output}\}$.

The configuration of the boolean formula will be included in the input, following the format mentioned previously. The rest of the variables have initial values of 0 except for $u_i, 1 \leq i \leq n$ and $synch$ which have initial values of 1.

Computation then proceeds as follows: Similar to the solutions to the Subset Sum problem, the program $p_{1,1}$, occasionally with the help of program $p_{2,1}$ will generate a binary string of length n. In this problem, the binary string generated will serve as a guide as to whether to add another value of 1 or 0 to T_i.

Since only variables $u_i, 1 \leq i \leq n$ and $synch$, as well as the input variables have initial values greater than or equal to the threshold 1, only programs of type $p_{1,1}$ and $p_{5,1}$, and $p_{11,1}$ and $p_{12,1}$ are applicable at the start. The program $p_{5,1}$ gives a value of 1 to all $T_i, 1 \leq i \leq n$ and replenishes the value of $synch$. The first binary string will be generated as well. $k_{j,i}$ and $\overline{k_{j,i}}$ will also have a value depending on their corresponding literal from the clause, 1 for $true$, 0 for $false$, and min_j will contain the minimum value for each clause to have a true value. At the second step, $p_{4,1}$ is now active along with the programs of the previous steps. This program adds the 1 from t_j to T_j. This determines whether T_i will have a value of 2 ($TRUE$) or 1 ($FALSE$). (Since for this problem 1 = $false$ and 2 = $true$, the minimum value min_j for every clause is simply the number of literals in the clause. If at least one of the literals have a value of 2, the whole clause will have a value of *true*.)

The T_j's will now have values greater than or equal to the threshold. Thus at the third step, programs of type $p_{6,1}$ and $p_{7,1}$ are applicable. The former program, on the one hand, represents copying the values of T_j to each C_j (representative of the jth clause). $p_{4,1}$, on the other hand, flips these values (from 2 to 1, or from 1 to 2) and passes to each C_j. At the end of this step, all C_j's have values greater than or equal to the threshold.

At the fourth step, programs of type $p_{8,1}$ are applied. If the value of C_j is greater than min_j, V_j will have a value greater than or equal to 1. Should all of the clauses evaluate to $TRUE$, $p_{9,1}$ will make the values of *output* and h equal to 1. If, however, at least one of the clauses evaluates to $FALSE$, $p_{10,1}$ will reset the values of all V_j, and the process is repeated from the second step onwards. If the values of $t_1, t_2, \cdots, t_n$ are all 1's but a satisfying assignment has not been found, the program $p_{3,1}$ becomes applicable, adding a value of 1 to h. At the next step, rules $p_{16,1}$ up to $p_{19,1}$ are applied, eventually rendering programs $p_{1,1}$, $p_{2,1}$ and $p_{5,1}$ inactive.

Regardless of output, the computation stops one step after the variable h gets a value of 1.

Corollary 3. *There exists a family of deterministic LTNP systems solving* **SAT** *having 1 membrane,* $4nk + 3n + 3k + 4$ *variables,* $8nk + 2n + 4k + 4 + \frac{n(n-1)}{2}$ *programs with polynomial production functions of* n *and* k*, working in* $allP$ *application mode, that runs in at most* $2^n + 6$ *steps where* n *is the number of boolean variables and* k *is the number of clauses in* ϕ.

6.2 A solution to `Boolean Satisfiability Problem` using LTNP in polynomial time

The construct of the deterministic LTNP system solving an instance $I = \phi$ of **SAT** having n boolean variables and m clauses is as follows:

$\Pi_I^D = (1, \{1\}, [_1]_1, 1, (Var_1, Pr_1, Var_1(0)), Var_{in}, Var_{out})$
where

– $Var_1 = \{var_{j,i}, 1 \leq j \leq m, 1 \leq i \leq n\} \cup$
$\{\overline{var_{j,i}}, 1 \leq j \leq m, 1 \leq i \leq n\} \cup \{v_i, 1 \leq i \leq n\} \cup$
$\{\overline{v_i}, 1 \leq i \leq n\} \cup \{T_{p,i}, 1 \leq p \leq 2^{n-1}, 1 \leq i \leq n\} \cup$
$\{X_{p,i}, 1 \leq p \leq 2^{n-1}, 1 \leq i \leq n\} \cup \{V_{p,j}, 1 \leq p \leq 2^{n-1}, 1 \leq j \leq m\} \cup$
$\{C_j, 1 \leq j \leq m\} \cup \{k_{j,i}, 1 \leq j \leq m, 1 \leq i \leq n\} \cup$
$\{\overline{k_{j,i}}, 1 \leq j \leq m, 1 \leq i \leq n\} \cup \{min_j, 1 \leq j \leq m\} \cup \{b, h, g, synch, \mathsf{output}\}$;

– Pr_1 is composed of programs of the following forms:
- $p_{1,1} : b + 1|_1 \rightarrow 1|b$,
- $p_{2,1} : (b - i + 1)(i - b + 1)|_1 \rightarrow 1|v_i, 1 \leq i \leq n$,
- $p_{3,1} : -(b - i + 1)(i - b + 1)|_1 \rightarrow 1|\overline{v_i}, 1 \leq i \leq n$,

- $p_{4,1} : \overline{v_i}|_1 \rightarrow 1|v_i, 1 \leq i \leq n,$

- $p_{5,1} : 3(v_i)(X_{p,i})|_1 \rightarrow 1|X_{2p,i} + 1|X_{2p,i+1} + 1|X_{2p-1,i+1},$
  $1 \leq i \leq n$ and $1 \leq p \leq 2^{n-1},$
- $p_{6,1} : 2(v_i)(X_{p,i})(X_{p,j})|_1 \rightarrow 1|X_{2p-1,j} + 1|X_{2p,j},$
  $j < i, 1 \leq i \leq n,$ and $1 \leq p \leq 2^{n-1},$

- $p_{7,1} : (X_{p,1})(X_{p,2}) \cdots (X_{p,n})|_1 \rightarrow 1|h, 1 \leq p \leq 2^{n-1}$
- $p_{8,1} : -(b + h)|_1 \rightarrow 1|b$

- $p_{9,1} : X_{p,i}|_1 \rightarrow 1|T_{p,i}, 1 \leq i \leq n$ and $1 \leq p \leq 2^{n-1},$
- $p_{10,1} : [(2^{n-1})(n)+1](synch)|_1 \rightarrow 1|T_{1,1}+1|T_{1,2}+\cdots+1|T_{2^{n-1},n}+1|synch,$
- $p_{11,1} : (k_{j,i})(T_{p,i})|_1 \rightarrow 1|C_{p,j},$
  $1 \leq j \leq m, 1 \leq i \leq n,$ and $1 \leq p \leq 2^{n-1},$
- $p_{12,1} : (\overline{k_{j,i}})(-T_{p,i} + 3)|_1 \rightarrow 1|C_{p,j}$
  $1 \leq j \leq m, 1 \leq i \leq n,$ and $1 \leq p \leq 2^{n-1},$

- $p_{13,1} : C_{p,j} - min_j|_1 \rightarrow 1|V_{p,j}, 1 \leq j \leq m,$ and $1 \leq p \leq 2^{n-1},$
- $p_{14,1} : 2(V_{p,j})(V_{p,j-1}) \cdots (V_{p,1})|_1 \rightarrow 1|\mathsf{output} + 1|h,$
  $1 \leq j \leq m,$ and $1 \leq p \leq 2^{n-1},$
- $p_{15,1} : V_{p,j}|_1 \rightarrow 1|g, 1 \leq j \leq m,$ and $1 \leq p \leq 2^{n-1},$

- $p_{16,1} : 2(var_{j,i})|_1 \rightarrow 1|\underline{k_{j,i}} + 1|min_j,\ 1 \leq j \leq m$ and $1 \leq i \leq n,$
- $p_{17,1} : 2(\overline{var_{j,i}})|_1 \rightarrow 1|\overline{k_{j,i}} + 1|min_j, 1 \leq j \leq m$ and $1 \leq i \leq n,$
- $p_{18,1} : \underline{k_{j,i}}|_1 \rightarrow 1|\underline{k_{j,i}}, 1 \leq j \leq m$ and $1 \leq i \leq n,$
- $p_{19,1} : \overline{k_{j,i}}|_1 \rightarrow 1|\overline{k_{j,i}}, 1 \leq j \leq m$ and $1 \leq i \leq n,$
- $p_{20,1} : min_j|_1 \rightarrow 1|min_j, 1 \leq j \leq m,$

- $p_{21,1} : h(-\underline{k_{j,i}})|_1 \rightarrow 1|\underline{k_{j,i}}, 1 \leq j \leq m$ and $1 \leq i \leq n,$
- $p_{22,1} : h(-\overline{k_{j,i}})|_1 \rightarrow 1|\overline{k_{j,i}}, 1 \leq j \leq m$ and $1 \leq i \leq n,$
- $p_{23,1} : h(-min_j)|_1 \rightarrow 1|min_j, 1 \leq j \leq m,$
- $p_{24,1} : h(-synch)|_1 \rightarrow 1|synch;$

– $Var_1(0) = (var_{1,1}, var_{1,2}, \overset{nm}{\cdots}, var_{m,n}, \overline{var_{1,1}}, \overline{var_{1,2}}, \overset{nm}{\cdots}, \overline{var_{m,n}}, 0, \overset{n}{\cdots}, 0,$
$0, \overset{n}{\cdots}, 0, 0, \overset{(2^{n-1})(n)}{\cdots}, 0, 0, \overset{(2^{n-1})(n)}{\cdots}, 1, 0, \overset{(2^{n-1})(m)}{\cdots}, 0, 0, \overset{m}{\cdots}, 0, 0, \overset{mn}{\cdots}, 0,$
$0, \overset{mn}{\cdots}, 0, 0, \overset{m}{\cdots}, 0, 1, 0, 0, 1, 0)$
– $Var_{in} = \{Var_{j,i}, \overline{Var_{j,i}}, 1 \leq j \leq m, 1 \leq i \leq n\};$
– $Var_{out} = \{\mathsf{output}\}.$

where n is the number of variables (length of input), m is the number of clauses.

The variables b, $X_{1,1}$ and $synch$ are equal to 1. The values of variables $Var_{j,i}$ and $\overline{Var_{j,i}}$ come from the input. The rest of the variables are equal to 0.

The following is the sequence of processes:

In the first step, $p_{16,1}$ and $p_{17,1}$ fire. These rules add the values of the input to $k_{j,i}$, $\overline{k_{j,i}}$ and min_j. If $k_{j,i} = 1$, then the literal X_i is present in clause j. If

$\overline{k_{j,i}} = 1$, then the literal $\overline{X_i}$ is present in clause j. min_j represent the minimum value needed to make clause j have a value of $TRUE$. This pair of rules do not fire again for the remainder of the process.

Also in the first step, programs $p_{1,1}$ to $p_{3,1}$ fire. This goes on until the whole process is halted. $p_{1,1}$ simply increments to b. $p_{2,1}$ puts a value in v_i such that $v_i = 1$ when $b = i$ and $v_i \leq 0$ otherwise. $p_{3,1}$ adds the negative of v_i to $\overline{v_i}$. $p_{9,1}$ also fires, adding the value of $X_{p,i}$ to $T_{p,i}$ (in the first step, this means $X_{1,1}$ to $T_{1,1}$). $p_{10,1}$ adds 1 to all $T_{p,i}$ and to *synch*. For this particular system, $1 = FALSE$ and $2 = TRUE$.

In the next step, programs $p_{4,1}$ and $p_{5,1}$ (and in other iterations $p_{6,1}$) fire. $p_{11,1}$ and $p_{12,1}$ also fire. $p_{4,1}$ adds the value of $\overline{v_i}$ to v_i. Notice that $v_i \leq 0$ for all $i \neq b$ after this rule. Rules $p_{5,1}$ and $p_{6,1}$ are responsible for the evolution and multiplication of input. Refer to Section 4.2 for the details of this. $p_{11,1}$ adds the product of $k_{j,i}$ and $T_{p,i}$ to $C_{p,j}$, while $p_{12,1}$ adds the product of $\overline{k_{j,i}}$ and $(-T_{p,i} + 3)$ to $C_{p,j}$. $C_{p,j}$ represents the sums of the values for each clause. Notice that $(-T_{p,i} + 3)$ negates the truth value of $T_{p,i}$.

Then, $p_{13,1}$ fires, which adds the difference between $C_{p,j}$ and min_j to $V_{p,j}$. Note that $V_{p,j}$ will only have a positive value if $C_{p,j} > min_j$, and that $V_{p,j}$ will never have a negative value due to the fact that the minimum value of $C_{p,j}$ is min_j.

$p_{14,1}$ will run if and only if all the terms in $(V_{p,j})(V_{p,j-1}) \cdots (V_{p,1})$ for at least one p is equal to 1. This translates to one permutation of inputs resulting to a *true* value in all clauses. Note that this makes $output \geq 1$ and $h \geq 1$.

$p_{18,1}$, $p_{19,1}$, and $p_{20,1}$ are "maintenance" rules, whose purpose are to keep the values of $k_{j,i}$, $\overline{k_{j,i}}$ and min_j constant (and greater than 0). This ensures that the process will continue until halted.

$p_{7,1}$ fires when the string $11 \cdots 1$ is encountered, signifying that all possible permutations of input has been produced. The variable h gets a value greater than or equal to 1.

$p_{8,1}$, $p_{21,1}$, $p_{22,1}$, $p_{23,1}$, and $p_{24,1}$ fire only when $h = 1$, which signifies *halt*. This makes the values of *synch*, $k_{j,i}$, $\overline{k_{j,i}}$, min_j, and b equal to or less than 0, ensuring that no infinite loop occurs. This is the final step, regardless of output.

Corollary 4. *There exists a family of deterministic LTNP systems solving* **SAT** *having 1 membrane,* $(2^{n-1})(2n+k)+4nk+2n+2k+5$ *variables,* $(2^{n-1})[\frac{n(n-1)}{2} + 2nk + 2n + 2k + 2] + 6nk + 3n + 2k + 4$ *programs with polynomial production functions of* n *and* k*, working in allP application mode, that runs in at most* $n + 7$ *steps where* n *is the number of boolean variables and* k *is the number of clauses in* ϕ*.*

7 Final Remarks

In this paper was introduced possible techniques in solving computationally hard problems with Numerical P Systems with Thresholds. These involve producing all permutations until a correct one is found, or none at all.

This technique was used to provide deterministic families of solutions to two of these computationally hard problems, namely the `Subset Sum Problem` and the `Boolean Satisfiability Problem`. In one of these problems, the SAT, we introduced one way of representing a boolean formula in a way that can be used for Numerical P Systems.

One of the techniques mentioned, namely the Linear time Binary String Generator, is able to produce all possible permutations of a string with a given length - in linear time. This involved producing more than one strings in a single time step, and thus required more than one verifiers. As expected, this resulted in the explosion in the number of variables and programs.

One interesting question would be: how can we reduce the amount of resources that was used by this technique (number of variables, number of programs)? Notice that the LBSG, despite having a polynomial runtime, will still require exponential time during construction (as it uses exponentially many variables and programs). Additional further work can also include exploring other techniques in solving computationally hard problems deterministically. Could one be made that is more efficient in terms of complexity? Others may also utilize the given techniques in this paper to solve more problems.

Another major improvement would be to introduce uniformity to polynomial time binary string generators. This may involve modifying the existing generator, or even creating NPT systems entirely different from the ones introduced here.

It must also be noted that all the systems used in this paper are using a single membrane. While the Numerical P Systems used here do not have special properties for the skin of membranes, it may be interesting to explore the possible techniques in using more than one membranes. What advantages can be gained from this? A good example of this would be by using Numerical P systems with migrating variables, introduced in [15], where variables are no longer confined in a membrane and can migrate from one membrane to another.

Acknowledgments

N.H. S. Hernandez is supported by the UP Research Dissemination Grant 2016-Q4-22 and the Vea Family Technology for All professorial chair. F.G.C. Cabarle is grateful for a Faculty Research Incentive Award (2016–2017) from the College of Engineering, UP Diliman, the PhDIA Project No. 161606 from the UP Diliman, OVCRD, an RLC grant 2016–2017 also from OVCRD, and the Soy and Tess Medina Professorial Chair Award for 2016–2017.

References

1. The P systems web page, `http://ppage.psystems.eu/`.

2. Buiu, C., Vasile, C., Arsene, O.: Development of membrane controllers for mobile robots. Inf. Sci. 187, 33–51 (Mar 2012)
3. Cabarle, F.G., Hernandez, N.H., Martinez-del Amor, M.A.: Spiking Neural P Systems with Structural Plasticity: Attacking the Subset Sum Problem. In: Rozenberg, G., Salomaa, A., Sempere, J.M., Zandron, C. (eds.) Lecture Notes in Computer Science. vol. 9504, pp. 106–116. *Membrane Computing 16th International Conference, CMC 2015, Valencia, Spain, August 17-21, 2015, Revised Selected Papers*, Springer (2015)
4. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1979)
5. Hernandez, N.H., Cabarle, F.G.: Solving some Computationally Hard Problems using Numerical P Systems with Thresholds. Pre-proc. 5th Asian Conference on Membrane Computing (ACMC2016), 14 to 16 November 2016, University Kebangsaan, Malaysia. (2016)
6. Leporati, A., Mauri, G., Zandron, C., Păun, G., Pérez-Jiménez, M.J.: Uniform Solutions to SAT and Subset Sum by spiking neural P systems. Natural Computing 8(4), 681–702 (2009)
7. Păun, G., Păun, R.: Membrane computing and economics: Numerical P Systems. Fundamenta Informaticae 73(1), 213–227 (2006)
8. Păun, G.: Membrane Computing: An Introduction. Springer Berlin Heidelberg (2002)
9. Păun, G.: From Cells to (Silicon) Computers, and Back. Springer New York, New York, NY (2008)
10. Paun, G.: Membrane computing: History and brief introduction. In: Gelembe, E., Kahane, J.P. (eds.) Fundamental Concepts in Computer Science. pp. 17–41. Imperial College Press (2009)
11. Pérez-Jiménez, M.J.: A computational complexity theory in membrane computing. Membrane Computing LNCS 5957, 125–148 (2010)
12. Pérez-Jiménez, M., Riscos-Núñez, A.: Solving the Subset-Sum Problem by P Systems with Active Membranes. New Generation Computing 23(4), 367–384 (2005)
13. Vasile, C.I., Pavel, A.B., Dumitrache, I.: Universality of Enzymatic Numerical P systems. International Journal of Computer Mathematics 90(4), 869–879 (2013)
14. Zhang, Z., Pan, L.: Numerical P Systems with Thresholds. International Journal of Computers Communications and Control 11(2), 292–304 (2016)
15. Zhang, Z., Wu, T., Păun, A., Pan, L.: Numerical p systems with migrating variables. Theoretical Computer Science 641, 85 – 108 (2016)

A Literate Programming Pearl in cP Systems

Radu Nicolescu

Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand
`r.nicolescu@auckland.ac.nz`

Abstract. We assess the "computer science" capabilities of our cP systems by solving a version of a famous *programming pearl*, originally posed by Jon Bentley (1984): printing the most common words in a text file, in their frequency order. Several interesting solutions have been proposed by Knuth (an exquisite model of literate programming, 1986), McIlroy (an engineering example of combining a timeless set of tools, 1986), Hanson (an alternate efficient solution, 1987). Here we propose a concise efficient solution based on the fast parallel and associative capabilities of cP systems. We also briefly check their sorting capabilities and propose a dynamic version of the classical pigeonhole algorithm.

Keywords: Literate programming, sorting, parallel sorting, pigeonhole algorithm, dynamic pigeonhole algorithm, associative data structures, membrane computing, P systems, cP systems, inter-cell parallelism, intra-cell parallelism, Prolog terms and unification, complex symbols, nested subcells, generic rules.

1 Introduction and Background

We are further assessing our current version of P systems with complex symbols, called cP systems [7].

In a nutshell, cP systems make extensive use of embedded (nested) subcells, but – in contrast to traditional P systems – these subcells are data-only membranes (totally devoided of any own rules). To compensate this apparent lack, the top cells' rules extend traditional multiset rewriting rules by Prolog-like unifications. The net result is a powerful system which can crisply and efficiently solve complex problems in a large variety of domains.

In particular, cP systems enable a reasonably straightforward creation and manipulation of high-level data structures which are typical in high-level languages, such as: numbers, relations (graphs), associative arrays, lists, trees, strings.

Here we assess the "computer science" capabilities of the cP systems by solving a version of a famous *programming pearl*, posed by Jon Bentley (1984): printing the most common words in a text file, more precisely (but still a bit vague) [1]:

> Given a text file and an integer k, print the k most common words in the file (and the number of their occurrences) in decreasing frequency.

Additionally, the integer N is typically used for the number of words, d is the number of distinct words, and f is the highest frequency count. Of course, one typically assumes that $N > d > k$ and $N - d + 1 \geq f \geq N/d$, but some solutions are optimised for the more special case $N \gg d \gg k$.

Several interesting solutions have been proposed by Knuth in 1986 – an exquisite model of literate programming [1], McIlroy in 1986 – an engineering example of combining a timeless set of tools [1], Hanson in 1987 – an alternate efficient solution [12]. All these three solutions can be considered as great literate programming sample models, if we take "literal programming" in a generic sense – not just Knuth's WEB/TANGLE implementation [2].

Here we propose a concise efficient solution, following Hanson's revised formulation [12] of the original problem specification, which clarifies the slight ambiguity of the original:

> Given a text file and an integer k, you are to print the words (and their frequencies of occurrence) whose frequencies of occurrence are among the k largest in order of decreasing frequency.

A tiny but artificial example may clarify these specifications. Assume that $k = 2$ and the input text is:

```
ccc aa aa aa ccc bb d aa d
```

Note that, here, $N = 9, d = 4, f = 4$. Bentley's original formulation, used by Knuth and McIlroy [1], essentially requires – a bit ambiguously – one of the following two outputs:

```
4 aa
2 ccc
```

or

```
4 aa
2 d
```

In contrast, Hanson's revised formulation [12], requires the following output – which is unambiguous, if the order of word sublists is not relevant (i.e. ccc d $\equiv$ d ccc):

```
4 aa
2 ccc d
```

Schematically, all these there solutions follow *four main phases*: (I) reading and splitting the text file into words (parsing it); (II) computing the word frequencies; (III) sorting according to frequencies; and (IV) printing the required output.

Knuth and Hanson provide large *monolithic* solutions, which include all four phases. Moreover, they combine phases I and II, by using associative data structures: Knuth uses a custom hash-trie and Hanson a custom hashtable with splay (move to front) lists. For phase III, both authors try to use efficient sorting methods. Knuth uses a fast sorting method, assuming that $N \gg d \gg k$ and that most frequent words tend to appear from the beginning of the text – as McIlroy points out, this does not always hold. Hanson offers a more universal fast sorting method based on the *pigeonhole algorithm*, with f holes.

McIlroy's solution is a textbook example for the *separation-of-concerns* principle, via a pipeline of staple general-purpose utilities initially developed for UNIX. Each of the four phases is implemented via just one or two commands. Together, phases II and III take exactly three lines in the pipe [1]:

```
(3) sort |
(4) uniq -c |
(5) sort -rn |
```

Line (3) sorts the N input words (lexicographically). Line (4) counts then discards the duplicates, keeping d unique exemplars and their frequency counts (as count/word pairs). Line (5) sorts d count/word pairs, in reverse count order (numerically).

Intentionally not given here are pipe lines (1), (2) and (6), which deal with phases I and IV. Reading, splitting into words and printing can be defined in a seemingly endless multiplicity of ways, which may not be worth discussing here. In particular, the concept of "word" itself may be highly interpretable: does it include ASCII letters, UNICODE letters, digits, punctuation signs, does it have a length limit, etc. Here, we will stay away from this discussion.

McIlroy's solution is also reasonably fast - not as fast as the other two – but it is extremely crisp and clear, and can be flexibly adapted to other input and output formats. Such a solution can be developed and deployed in just a few minutes – this sounds amazing, but does not account for the many man-months required to develop and tune the used building blocks (UNIX tools). McIlroy also notes that his solution could be sped up by replacing the more costlier lines (3) and (4) by a hypothetical tool based on associative arrays – in fact, this would bring his solution closer to Hanson's solution for phase II.

Our cP solution – which uses *one single cell with data-only subcells* – follows the spirit of McIlroy's and Hanson's solutions. It is based on associative data types and a sorting idea close to Hanson's pigeonhole algorithm. It also uses a small number of rules – close to McIlroy's pipeline size – but, in contrast, it is built from scratch (not on higher building block as the UNIX commands).

We offer two alternate solutions: (i) a solution which solves Hanson's version of the problem – where the result is a *sorted sequence of word multisets*; and (ii) a solution which solves the original problem, as posed by Bentley and used by Knuth and McIlroy – where the result is a *sorted sequence of words.*

In this process, we propose and use a *dynamic pigeonhole algorithm*, adaptable to other platforms with strong associative capabilities, where – metaphorically - pigeonholes are only opened one at at time, instantly attracting objects with matching keys.

In our case, we must first adapt the above problem formulation to typical P systems, where cells contain multisets of symbols, not ordered structures.

What is a sorted multiset? Ordered structures must be constructed in terms of multisets – in cP systems, we can create the required high-level structures by deep nesting of complex symbols (subcells).

As above mentioned, we chose to skip over the reading phase (I) and we assume that all words are "magically" present at start-time in our single cell. Our focus is on phases II and III, where all operations are clearly defined and can be efficiently performed by cP systems.

Finally – as used in our first solution (i) – we simulate the printing phase IV, by *sequentially sending out the required results, in order, over a designated line.* Alternatively – as used in our second solution (ii) – we actually *build an ordered list containing the required results.*

For completeness, Section 2 introduces high-level data structures in cP systems and Appendix A offers a more complete definition of the cP systems – both these sections are largely reproduced from our earlier paper [7]. The remaining sections discuss our solution.

2 Data structures in cP systems

We assume that the reader is familiar with the membrane extensions collectively known as *complex symbols*, proposed by Nicolescu et al. [8, 9, 6]. However, to ensure some degree of self-containment, our revised extensions, called cP systems, are reproduced in Appendix A.

In this section we sketch the design of high-level data structures, similar to the data structures used in high-level pseudocode or high-level languages: numbers, relations, functions, associative arrays, lists, trees, strings, together with alternative more readable notations.

Natural numbers. Natural numbers can be represented via *multisets* containing repeated occurrences of the *same* atom. For example, considering that *1* represents an ad-hoc unary digit, the following complex symbols can be used to describe the contents of a virtual integer *variable* a: $a() = a(\lambda)$ — the value of a is 0; $a(1^3)$ — the value of a is 3. For concise expressions, we may alias these number representations by their corresponding numbers, e.g. $a() \equiv a(0), b(1^3) \equiv b(3)$. Nicolescu et al. [8, 9] show how the basic arithmetic operations can be efficiently modelled by P systems with complex symbols.

Relations and functions. Consider the *binary relation* r, defined by: $r = \{(a, b),\ (b, c),\ (a, d),\ (d, c)\}$ (which has a diamond-shaped graph). Using complex symbols, relation r can be represented as a *multiset* with four r items, $\{r(\kappa(a)\ \upsilon(b)),\ r(\kappa(b)\ \upsilon(c)),\ r(\kappa(a)\ \upsilon(d)),\ r(\kappa(d)\ \upsilon(c))\}$, where ad-hoc atoms κ and υ introduce *domain* and *codomain* values (respectively). We may also alias the items of this multiset by a more expressive notation such as: $\{(a \overset{r}{\rightleftarrows} b),\ (b \overset{r}{\rightleftarrows} c),\ (a \overset{r}{\rightleftarrows} d),\ (d \overset{r}{\rightleftarrows} c)\}$.

If the relation is a *functional relation*, then we can emphasise this by using another operator, such as "mapsto". For example, the functional relation $f =$

$\{(a,b),\ (b,c),\ (d,c)\}$ can be represented by multiset $\{f(\kappa(a)\ \upsilon(b)),\ f(\kappa(b)\ \upsilon(c)),\ f(\kappa(d)\ \upsilon(c))\}$ or by the more suggestive notation: $\{(a \overset{f}{\mapsto} b),\ (b \overset{f}{\mapsto} c),\ (d \overset{f}{\mapsto} c)\}$. To highlight the actual mapping value, instead of $a \overset{f}{\mapsto} b$, we may also use the succinct abbreviation $f[a] = b$.

In this context, the $\rightleftarrows$ and $\mapsto$ operators are considered to have a high associative priority, so the enclosing parentheses are mostly used for increasing the readability.

Associative arrays. Consider the *associative array* x, with the following key-value mappings (i.e. functional relation): $\{1 \mapsto a; 1^3 \mapsto c; 1^7 \mapsto g\}$. Using complex symbols, array x can be represented as a multiset with three items, $\{x(\kappa(1)\,\upsilon(a)),\ x(\kappa(1^3)\,\upsilon(c)),\ x(\kappa(1^7)\,\upsilon(g))\}$, where ad-hoc atoms κ and υ introduce keys and values (respectively). We may also alias the items of this multiset by the more expressive notation $\{1 \overset{x}{\mapsto} a,\ 1^3 \overset{x}{\mapsto} c,\ 1^7 \overset{x}{\mapsto} g\}$.

Lists. Consider the *list* y, containing the following sequence of values: $[u; v; w]$. List y can be represented as the complex symbol $y(\ \gamma(u\ \gamma(v\ \gamma(w\ \gamma()))))$, where the ad-hoc atom γ represents the list constructor *cons* and $\gamma()$ the empty list. We may also alias this list by the more expressive equivalent notation $y(u \,|\, v \,|\, w)$ – or by $y(u \,|\, y')$, $y'(v \,|\, w)$ – where operator $|$ separates the head and the tail of the list. The notation $z(|)$ is shorthand for $z(\gamma())$ and indicates an empty list, z.

Trees. Consider the *binary tree* z, described by the structured expression $(a, (b), (c, (d), (e)))$, i.e. z points to a root node which has: (i) the value a; (ii) a left node with value b; and (iii) a right node with value c, left leaf d, and right leaf e. Tree z can be represented as the complex symbol $z(a\ \phi(b)\ \psi(c\ \phi(d)\ \psi(e)))$, where ad-hoc atoms ϕ, ψ introduce left subtrees, right subtrees (respectively).

Strings. Consider the *string* $s =$ "*abc*", where a, b, and c are atoms. Obviously, string s can interpreted as the list $s = [a; b; c]$, i.e. string s can be represented as the complex symbol $s(\ \gamma(a\ \gamma(b\ \gamma(c\ \gamma()))))$, etc.

3 The parallel cP algorithm – solution (i)

3.1 Initial state

We need *one single cell* with one designated output line. Required data structures are built as complex symbols (data-only subcells), using the interpretations and notations defined in Section 2. In particular, the N input words are strings built via functor w; these complex symbols are already extant when the systems starts. Figure 1 illustrates the initial cell contents for the sample given in Section 1.

3.2 Phase II

Using an associative relation, α, each word is tagged with an initial "frequency" count of 1 and then we *merge all word duplicates* and *sum* their associated counts. In the end, we get d words, each one with its actual frequency count.

“*ccc*” “*aa*” “*aa*” “*aa*” “*ccc*” “*bb*” “*d*” “*aa*” “*d*”

(a) High-level strings.

$$w(c\,w(c\,w(c\,w())))\quad w(a\,w(a\,w()))\quad w(a\,w(a\,w()))\quad w(a\,w(a\,w()))$$
$$w(c\,w(c\,w(c\,w())))\quad w(b\,w(b\,w()))\quad w(d\,w())\quad w(a\,w(a\,w()))\quad w(d\,w())$$

(b) Underlying complex symbols.

Fig. 1: Sample initial word multiset.

Figure 2 shows the three rules for phase II. This ruleset starts in state S_0. Rule (0) establishes relation α between extant strings given by $w(X)$ and the initial frequency count 1; it runs in `max` mode, so it completes its job in 1 cP step.

Rule (1) repeatedly merges word duplicates and sums their associated counts; it runs in `max` mode, so it completes its job in $\log(d)$ cP steps – this rule is non-deterministic but confluent.

After rule (1) completes, rule (2) moves to the final state of this ruleset, S_2. Table 1 illustrates the evolution of the cell contents for our initial sample.

$$\begin{array}{llllr}
S_0 & w(W) & \rightarrow_{\texttt{max}\otimes\texttt{min}} & S_1\ \alpha(w(W)\,f(1)) & (0)\\
S_1 & \alpha(w(W)\,f(F))\ \ \alpha(w(W)\,f(F')) & \rightarrow_{\texttt{max}\otimes\texttt{min}} & S_1\ \alpha(w(W)\,f(FF')) & (1)\\
S_1 & & \rightarrow_{\texttt{min}\otimes\texttt{min}} & S_2 & (2)
\end{array}$$

Fig. 2: Ruleset for phase II.

3.3 Phase III

We create maximal word multisets by merging all words sharing the same *frequency counts.*

Figure 3 shows the two rules for phase III. This ruleset starts in state S_2, the final state for phase II (3.2). Rule (3) merges word multisets sharing the same frequency counts; it runs in `max` mode, so it completes its job in $\log(f)$ cP steps – this rule is non-deterministic but confluent.

Apply	State	Cell contents
rule (0)	S_0	"*ccc*" "*aa*" "*aa*" "*aa*" "*ccc*" "*bb*" "*d*" "*aa*" "*d*"
rule (1)	S_1	$\alpha($"*ccc*"$f(1))$ $\alpha($"*aa*"$f(1))$ $\alpha($"*aa*"$f(1))$ $\alpha($"*aa*"$f(1))$ ...
rule (1)	S_1	$\alpha($"*ccc*" $f(2))$ $\alpha($"*aa*" $f(2))$ $\alpha($"*aa*" $f(2))$ $\alpha($"*bb*" $f(1))$ $\alpha($"*d*" $f(2))$
rule (2)	S_1	$\alpha($"*ccc*" $f(2))$ $\alpha($"*aa*" $f(4))$ $\alpha($"*bb*" $f(1))$ $\alpha($"*d*" $f(2))$
–	S_2	$\alpha($"*ccc*" $f(2))$ $\alpha($"*aa*" $f(4))$ $\alpha($"*bb*" $f(1))$ $\alpha($"*d*" $f(2))$

Table 1: Phase II evolution of the sample word multiset.

After rule (3) completes, rule (4) moves to the final state of this ruleset, S_3. Table 2 illustrates the evolution of the cell contents for the initial sample.

$$
\begin{array}{lllll}
S_2 & \alpha(W\,f(F)) \;\; \alpha(W'\,f(F)) & \rightarrow_{\texttt{max}\otimes\texttt{min}} & S_2\ \alpha(W\,W'\,f(F)) & (3) \\
S_2 & & \rightarrow_{\texttt{min}\otimes\texttt{min}} & S_3 & (4)
\end{array}
$$

Fig. 3: Ruleset for phase III.

Apply	State	Cell contents
rule (3)	S_2	$\alpha($"*ccc*" $f(2))$ $\alpha($"*aa*" $f(4))$ $\alpha($"*bb*" $f(1))$ $\alpha($"*d*" $f(2))$
rule (4)	S_2	$\alpha($"*ccc*" "*d*"$f(2))$ $\alpha($"*aa*" $f(4))$ $\alpha($"*bb*" $f(1))$
–	S_3	$\alpha($"*ccc*" "*d*"$f(2))$ $\alpha($"*aa*" $f(4))$ $\alpha($"*bb*" $f(1))$

Table 2: Phase III evolution of the sample word multiset.

3.4 Phase IV

We send out all existing word multisets, sequentially, in decreasing order of their *frequency counts*. We propose and use a *dynamic* version of the classical *pigeonhole algorithm* (adaptable to other platforms with strong associative capabilities), where – metaphorically - pigeonholes are only opened one at at time, instantly attracting objects with matching keys.

First, we determine the highest frequency count. Next, we repeatedly output the word multiset having the current highest frequency count – if any – and then decrement this count, until we reach 0. This current highest frequency count is the "enabled pigeonhole" which "attracts" the word multiset having the same frequency count. For simplicity, we do not consider the parameter k, but it is straightforward to include it in this ruleset.

Figure 4 shows the rules for phase IV. This ruleset starts in state S_3, the final state for phase III (3.3). Rule (5) extracts frequency counts; it runs in `max` mode, so it completes its job in 1 cP steps.

Rule (6) determines the highest frequency count by taking pairwise maximums (note that all extant frequency counts are different); it runs in `max` mode, so it completes its job in $\log(f)$ cP steps – this rule is non-deterministic but confluent.

After rule (6) completes, rule (7) moves to the next state of this ruleset, S_5. Rule (8) outputs the word multiset having the current (highest) non-zero frequency count – if any – and then decrements this count; rule (9) just decrements this count, if there is no matching word multiset; this pair of rules complete their job in $\log(f)$ cP steps.

After all the word multisets are sent out, the cell remains idle in the final state, S_5 – alternatively, one more rule could clear the remaining $f(0)$ counter and transit to another state (e.g. S_6). Table 3 illustrates the evolution of the cell contents for the initial sample. Essentially, in this scenario we output the sequence [("aa", 4); ("ccc" "d", 2); ("bb", 1)],

$$\begin{array}{llllr}
S_3 & \alpha(W\, f(F)) & \rightarrow_{\texttt{max}\otimes\texttt{min}} & S_4\ \alpha(W\, f(F))\ f(F) & (5) \\
S_4 & f(F)\ \ f(F1F') & \rightarrow_{\texttt{max}\otimes\texttt{min}} & S_4\ f(F1F') & (6) \\
S_4 & & \rightarrow_{\texttt{min}\otimes\texttt{min}} & S_5 & (7) \\
S_5 & \alpha(W\, f(F1))\ \ f(F1) & \rightarrow_{\texttt{min}\otimes\texttt{min}} & S_5\ \alpha(W\, f(F1)) \downarrow\ \ f(F) & (8) \\
S_5 & f(F1) & \rightarrow_{\texttt{min}\otimes\texttt{min}} & S_5\ f(F) & (9)
\end{array}$$

Fig. 4: Ruleset for phase IV.

4 The parallel cP algorithm – alternate solution (ii)

Here we sketch an alternate implementation, which actually builds a *sorted list of words*, ordered on their frequency counts. This solution could be applied to get a sorted list of word multisets, but here we use it to get a *sorted list of words*,

Apply	State	Cell contents
rule (5)	S_3	$\alpha(\text{“}ccc\text{” “}d\text{”} f(2))$ $\alpha(\text{“}aa\text{”} f(4))$ $\alpha(\text{“}bb\text{”} f(1))$
rule (6)	S_4	$\alpha(\text{“}ccc\text{” “}d\text{”} f(2))$ $\alpha(\text{“}aa\text{”} f(4))$ $\alpha(\text{“}bb\text{”} f(1))$ $f(2)$ $f(4)$ $f(1)$
rule (6)	S_4	$\alpha(\text{“}ccc\text{” “}d\text{”} f(2))$ $\alpha(\text{“}aa\text{”} f(4))$ $\alpha(\text{“}bb\text{”} f(1))$ $f(4)$ $f(1)$
rule (7)	S_4	$\alpha(\text{“}ccc\text{” “}d\text{”} f(2))$ $\alpha(\text{“}aa\text{”} f(4))$ $\alpha(\text{“}bb\text{”} f(1))$ $f(4)$
rule (8)	S_5	$\alpha(\text{“}ccc\text{” “}d\text{”} f(2))$ $\alpha(\text{“}\mathbf{aa}\text{”}\ \mathbf{f(4)})$ $\alpha(\text{“}bb\text{”} f(1))$ $f(4)$
rule (9)	S_5	$\alpha(\text{“}ccc\text{” “}d\text{”} f(2))$ $\alpha(\text{“}bb\text{”} f(1))$ $f(3)$
rule (8)	S_5	$\alpha(\text{“}\mathbf{ccc}\text{” “}\mathbf{d}\text{”}\mathbf{f(2)})$ $\alpha(\text{“}bb\text{”} f(1))$ $f(2)$
rule (8)	S_5	$\alpha(\text{“}\mathbf{bb}\text{”}\ \mathbf{f(1)})$ $f(1)$
–	S_5	$f(0)$

Table 3: Phase IV evolution of the sample word multiset – each time it is applied, the highlighted rule (8) outputs one word multiset and its associated frequency count.

i.e. a result closer to the original problem formulation posed by Bentley and used by Knuth and McIlroy [1].

Conceptually, we start from the interim results of phase II of solution (i) (3.2), but this time we give a complete solution (not explicitly split into phases).

We create a list of words, sorted in decreasing order of their *frequency counts*. As in the earlier phase II (3.2) each word is tagged with an initial “frequency” count of 1 and then we *merge all word duplicates* and *sum* their associated counts. In the end, we get d words, each one with its actual frequency count.

Then, as in the earlier phase IV (3.4), we use a *dynamic* version of the classical *pigeonhole algorithm*, but this time we stack the “attracted” words in a result list (instead of sending them out).

First, we “enable a pigeonhole” for frequency 1 and create an empty result list. Next, we repeatedly stack all words having the current pigeonhole frequency count – if any – and then increment this count, until we exhaust all extant words. For simplicity, we again do not consider the parameter k, but it is straightforward to include it in this ruleset.

Figure 5 shows all rules for this alternate solution. Rules (0) and (1) are exactly as in the earlier phase II. Rule (2) is modified: to “enable a pigeonhole” for frequency 1 and to create an empty result list, ρ.

Rule (3) repeatedly stacks onto ρ all words having the current frequency count – if any; the standalone f acts as a promoter. Rule (4) increments this frequency count, if there are no (more) matching words for this count, but there are still other words to process; any extant $\alpha(...)$ acts as a promoter. The rules pair (3) and (4) complete their job in $\log(f)$ cP steps.

After all the words are stacked, the cell remains idle in the final state, S_2. The evolution is non-deterministic, which exactly corresponds to the slight vagueness of the original problem formulation. Table 4 illustrates a possible evolution of the cell contents for the initial sample. Essentially, in this scenario we obtain the list [("aa", 4); ("d" 2); ("ccc" 2); ("bb", 1)], but we could have also obtained the list [("aa", 4); ("ccc" 2); ("d" 2); ("bb", 1)].

$$\begin{array}{llllll}
S_0 & w(W) & & \rightarrow_{\texttt{max}\otimes\texttt{min}} & S_1\ \alpha(w(W)\, f(1)) & (0) \\
S_1 & \alpha(w(W)\, f(F)) & \alpha(w(W)\, f(F')) & \rightarrow_{\texttt{max}\otimes\texttt{min}} & S_1\ \alpha(w(W)\, f(FF')) & (1) \\
S_1 & & & \rightarrow_{\texttt{min}\otimes\texttt{min}} & S_2\ f(1)\ \rho() & (2) \\
S_2 & \alpha(w(W)\, f(F)) & \rho(R) & \rightarrow_{\texttt{max}\otimes\texttt{min}} & S_2\ \rho(\alpha(w(W)\, f(F))\, \rho(R)) & (3) \\
 & & & & \quad \mid f(F) & \\
S_2 & f(F) & & \rightarrow_{\texttt{min}\otimes\texttt{min}} & S_2\ f(F1) & (4) \\
 & & & & \quad \mid \alpha(\_) &
\end{array}$$

Fig. 5: Ruleset for alternate solution (ii).

5 Reflections and open problems

Both our solutions seem to have an optimal *runtime complexity*, or close to it, essentially $\mathcal{O}(\log(d)+\log(f))$ cP steps, which, in the worst case, is $\mathcal{O}(\log(N))$, but typically is much smaller. This optimality is not proven, but seems a believable hypothesis.

Also, our solutions seem to have a very decent *static complexity*, comparable to the the best known solution in this regard, proposed by McIlroy: 10 or 5 rules – in our two solutions – vs. 4 lines – the combination of 4 powerful UNIX commands in McIlroy's excellent solution. Moreover, in contrast to this, our solutions are build from "scratch" (including the associative sorting!), not on other complex utilities. Also, as presented, McIlroy's solution runs in $\mathcal{O}(N \log(N))$ steps (because of the initial sorting), which makes it slower than ours. In all fairness, McIlroy mentions potential speed-ups, but these do not seem yet available.

In fact, these comparisons may be misleading, as our solution runs on a highly parallel engine – cP systems – while the other solutions are purely sequential. It may be interesting to evaluate other parallel solutions to this problem, including other P systems solutions, but we are not aware of any.

As earlier mentioned, cP systems rules generalise the traditional P systems rules by powerful Prolog-like unifications, but the classical Prolog unification

Apply	State	Cell contents
rule (0)	S_0	$“ccc”\ “aa”\ “aa”\ “aa”\ “ccc”\ “bb”\ “d”\ “aa”\ “d”$
rule (1)	S_1	$\alpha(“ccc” f(1))\ \ \alpha(“aa” f(1))\ \ \alpha(“aa” f(1))\ \ \alpha(“aa” f(1))\ \ \ldots$
rule (1)	S_1	$\alpha(“ccc”\ f(2))\ \ \alpha(“aa”\ f(2))\ \ \alpha(“aa”\ f(2))\ \ \alpha(“bb”\ f(1))\ \ \alpha(“d”\ f(2))$
rule (2)	S_1	$\alpha(“ccc”\ f(2))\ \ \alpha(“aa”\ f(4))\ \ \alpha(“bb”\ f(1))\ \ \alpha(“d”\ f(2))$
rule (3)	S_2	$f(1)\ \ \alpha(“ccc”\ f(2))\ \ \alpha(“aa”\ f(4))\ \ \alpha(“bb”\ f(1))\ \ \alpha(“d”\ f(2))\ \ \rho()$
rule (4)	S_2	$f(1)\ \ \alpha(“ccc”\ f(2))\ \ \alpha(“aa”\ f(4))\ \ \alpha(“d”\ f(2))\ \ \rho(\alpha(“bb”\ f(1))\ \ \rho())$
rule (3)	S_2	$f(2)\ \ \alpha(“ccc”\ f(2))\ \ \alpha(“aa”\ f(4))\ \ \alpha(“d”\ f(2))\ \ \rho(\alpha(“bb”\ f(1))\ \ \rho())$
rule (3)	S_2	$f(2)\ \ \alpha(“aa”\ f(4))\ \ \alpha(“d”\ f(2))\ \ \rho(\alpha(“ccc”\ f(2))\ \ \rho(\alpha(“bb”\ f(1))\ \rho()))$
rule (4)	S_2	$f(2)\ \ \alpha(“aa”\ f(4))\ \ \rho(\alpha(“d”\ f(2))\ \rho(\alpha(“ccc”\ f(2))\ \rho(\alpha(“bb”\ f(1))\ \rho())))$
rule (4)	S_2	$f(3)\ \ \alpha(“aa”\ f(4))\ \ \rho(\alpha(“d”\ f(2))\ \rho(\alpha(“ccc”\ f(2))\ \rho(\alpha(“bb”\ f(1))\ \rho())))$
rule (3)	S_2	$f(4)\ \ \alpha(“aa”\ f(4))\ \ \rho(\alpha(“d”\ f(2))\ \rho(\alpha(“ccc”\ f(2))\ \rho(\alpha(“bb”\ f(1))\ \rho())))$
–	S_2	$f(4)\ \ \rho(\alpha(“aa”\ f(4))\ \rho(\alpha(“d”\ f(2))\ \rho(\alpha(“ccc”\ f(2))\ \rho(\alpha(“bb”\ f(1))\ \rho()))))$

Table 4: Alternate solution (ii): possible evolution of the sample word multiset. Here the final result is the sorted list $[\alpha(“aa”\ f(4)); \alpha(“d”\ f(2)); \alpha(“ccc”\ f(2)); \alpha(“bb”\ f(1))]$.

algorithms do *not* work on multisets. More work is needed to design efficient unification algorithms which work on multisets and and scale out well on parallel architectures.

It is also interesting to note that our solutions seem to struggle a bit when they are constrained to run in a purely sequential mode, as in phase IV of solution (i), but feel more comfortable when they can unleash the parallel associative potential of cP systems, as in solution (ii).

To the best of our knowledge, this paper proposes a novel sorting algorithm, with a remarkable crisp expression: a dynamic version of the classical pigeonhole algorithm, apparently suitable for any platform with strong associative features (such as many or most versions of P systems).

Finally, as an open problem, it might be worthwhile to invest more effort into developing a real literate model for P systems and to develop a set of tools corresponding to Knuth's WEB toolset – perhaps P-WEB or cP-WEB?

References

1. Bentley, J., Knuth, D., McIlroy, D.: Programming pearls: A literate program. Commun. ACM 29(6), 471–483 (Jun 1986), `http://doi.acm.org/10.1145/5948.315654`

2. Knuth, D.E.: Literate programming. Comput. J. 27(2), 97–111 (May 1984), http://dx.doi.org/10.1093/comjnl/27.2.97
3. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996)
4. Nicolescu, R.: Parallel and distributed algorithms in P systems. In: Gheorghe, M., Păun, G., Rozenberg, G., Salomaa, A., Verlan, S. (eds.) Membrane Computing, CMC 2011, Revised Selected Papers, Lecture Notes in Computer Science, vol. 7184, pp. 35–50. Springer Berlin / Heidelberg (2012)
5. Nicolescu, R.: Parallel thinning with complex objects and actors. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) 15th International Conference, CMC 2014, Prague, Czech Republic, August 20-22, 2014, Revised Selected Papers, Lecture Notes in Computer Science, vol. 8961, pp. 330–354. Springer (2015)
6. Nicolescu, R.: Structured grid algorithms modelled with complex objects. In: Rozenberg, G., Salomaa, A., Sempere, J.M., Zandron, C. (eds.) 16th Conference on Membrane Computing (CMC16), Revised Selected Papers, Lecture Notes in Computer Science, vol. 9504, pp. 321–337. Springer (2015)
7. Nicolescu, R.: Revising the membrane computing model for byzantine agreement. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) 17th International Conference on Membrane Computing (CMC17), Revised Selected Papers, Lecture Notes in Computer Science, vol. 10105, pp. 317–339. Springer (2016)
8. Nicolescu, R., Ipate, F., Wu, H.: Programming P systems with complex objects. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y., Rozenberg, G., Salomaa, A. (eds.) 14th Conference on Membrane Computing, Revised Selected Papers, Lecture Notes in Computer Science, vol. 8340, pp. 280–300. Springer (2013)
9. Nicolescu, R., Wu, H.: Complex objects for complex applications. Romanian Journal of Information Science and Technology 17(1), 46–62 (2014)
10. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., New York, NY, USA (2010)
11. Tel, G.: Introduction to Distributed Algorithms. Cambridge University Press (2000)
12. Van Wyk, C.J.: Literate programming. Commun. ACM 30(7), 583–599 (Jul 1987), http://doi.acm.org/10.1145/28569.315738

A Appendix cP Systems : P Systems with Complex Symbols

We present the details of our complex-symbols framework, slightly revised from our earlier papers [5, 6].

A.1 Complex symbols as subcells

Complex symbols play the roles of cellular micro-compartments or substructures, such as organelles, vesicles or cytoophidium assemblies ("snakes"), which are embedded in cells or travel between cells, but without having the full processing power of a complete cell. In our proposal, *complex symbols* represent nested data compartments which have no own processing power: they are acted upon by the rules of their enclosing cells.

Technically, our *complex symbols*, also called *subcells*, are similar to Prolog-like *first-order terms*, recursively built from *multisets* of atoms and variables. *Atoms* are typically denoted by lower case letters (or, occasionally, digits), such as a, b, c, *1*. *Variables* are typically denoted by uppercase letters, such as X, Y, Z. For improved readability, we also consider *anonymous variables*, which are denoted by underscores ("_"). Each underscore occurrence represents a *new* unnamed variable and indicates that something, in which we are not interested, must fill that slot.

Terms are either (i) simple atoms, or (ii) atoms (called *functors*), followed by one or more parenthesized *multisets* (called *arguments*) of other symbols (terms or variables), e.g. $a(b^2X), a(X^2c(Y)), a(b^2)(c(Z))$. Functors that are followed by more than one parenthesized argument are called *curried* (by analogy to functional programming) and, as we see later, are useful to precisely described deep 'micro-surgical" changes which only affect inner nested symbols, without directly touching their enclosing outer symbols. Terms that do *not* contain variables are called *ground*, e.g.:

– Ground terms: a, $a(\lambda)$, $a(b)$, $a(bc)$, $a(b^2c)$, $a(b(c))$, $a(bc(\lambda))$, $a(b(c)d(e))$, $a(b(c)d(e))$, $a(b(c)d(e(\lambda)))$, $a(bc^2d)$; or, a curried form: $a(b^2)(c(d)e^3)$.
– Terms which are not ground: $a(X)$, $a(bX)$, $a(b(X))$, $a(XY)$, $a(X^2)$, $a(XdY)$, $a(Xc())$, $a(b(X)d(e))$, $a(b(c)d(Y))$, $a(b(X)d(e(Y)))$, $a(b(X^2)d(e(Xf^2)))$; or, a curried form: $a(b(X))(d(Y)e^3)$; also, using anonymous variables: $a(b\_)$, $a(X\_)$, $a(b(X)d(e(\_)))$.

Note that we may abbreviate the expression of complex symbols by removing inner λ's as explicit references to the empty multiset, e.g. $a(\lambda) = a()$.

Complex symbols (subcells, terms) can be formally defined by the following grammar:

```
<term> ::= <atom> | <functor> ( '(' <argument> ')' )+
<functor> ::= <atom>
<argument> ::= λ | ( <term-or-var> )+
<term-or-var> ::= <term> | <variable>
```

Unification. All terms (ground or not) can be (asymmetrically) *matched* against *ground* terms, using an ad-hoc version of *pattern matching*, more precisely, a *one-way first-order syntactic unification*, where an atom can only match another copy of itself, and a variable can match any bag of ground terms (including the empty bag, λ). This may create a combinatorial *non-determinism*, when a combination of two or more variables are matched against the same bag, in which case an arbitrary matching is chosen. For example:

– Matching $a(b(X)fY) = a(b(cd(e))f^2g)$ deterministically creates a single set of unifiers: $X, Y = cd(e), fg$.
– Matching $a(XY^2) = a(de^2f)$ deterministically creates a single set of unifiers: $X, Y = df, e$.

- Matching $a(XY) = a(df)$ non-deterministically creates one of the following four sets of unifiers: $X, Y = \lambda, df$; $X, Y = df, \lambda$; $X, Y = d, f$; $X, Y = f, d$.

Performance note. If the rules avoid any matching non-determinism, then this proposal should not affect the performance of P simulators running on existing machines. Assuming that bags are already taken care of, e.g. via hash-tables, our proposed unification probably adds an almost linear factor. Let us recall that, in similar contexts (no occurs check needed), Prolog unification algorithms can run in $O(ng(n))$ steps, where g is the inverse Ackermann function. Our conjecture must be proven though, as the novel presence of multisets may affect the performance.

A.2 Generic rules

Rules use *states* and are applied top-down, in the so-called *weak priority* order. Rules may contain *any* kind of terms, ground and not-ground. In *concrete* models, *cells* can only contain *ground* terms. *Cells* which contain *unground* terms can only be used to define *abstract* models, i.e. high-level patterns which characterise families of similar concrete models.

Pattern matching. Rules are matched against cell contents using the above discussed *pattern matching*, which involves the rule's left-hand side, promoters and inhibitors. Moreover, the matching is *valid* only if, after substituting variables by their values, the rule's right-hand side contains ground terms only (so *no* free variables are injected in the cell or sent to its neighbours), as illustrated by the following sample scenario:

- The cell's *current content* includes the *ground term*:
  $n(a\,\phi(b\,\phi(c)\,\psi(d))\,\psi(e))$
- The following *rewriting rule* is considered:
  $n(X\,\phi(Y\,\phi(Y_1)\,\psi(Y_2))\,\psi(Z)) \;\rightarrow\; v(X)\,n(Y\,\phi(Y_2)\,\psi(Y_1))\,v(Z)$
- Our pattern matching determines the following *unifiers*:
  $X = a$, $Y = b$, $Y_1 = c$, $Y_2 = d$, $Z = e$.
- This is a *valid* matching and, after *substitutions*, the rule's *right-hand* side gives the *new content*:
  $v(a)\;n(b\,\phi(d)\,\psi(c))\;v(e)$

Generic rules format. We consider rules of the following *generic* format (we call this format generic, because it actually defines templates involving variables):

$$\boxed{\begin{array}{lll} \textit{current-state}\;\; \textit{symbols}\ldots & \rightarrow_\alpha & \textit{target-state}\;\; (\textit{in-symbols})\ldots \\ & & (\textit{out-symbols})_\delta \ldots \\ & & \mid\; \textit{promoters}\ldots \;\; \neg\; \textit{inhibitors}\ldots \end{array}}$$

Where:

– All *symbols*, including *states*, *promoters* and *inhibitors*, are *multisets of terms*, possibly containing *variables* (which can be *matched* as previously described).
– Parentheses can be used to clarify the association of symbols, but otherwise have no own meaning.
– Subscript $\alpha \in \{\texttt{min}, \texttt{max}\} \times \{\texttt{min}, \texttt{max}\}$, indicates a combined *instantiation* and *rewriting* mode, as further discussed in the example below.
– *In-symbols* become available after the end of the current step only, as in traditional P systems (we can imagine that these are sent via an ad-hoc fast *loopback* channel);
– *Out-symbols* are sent, at the end of the step, to the cell's structural neighbours. These symbols are enclosed in round parentheses which further indicate their destinations, above abbreviated as δ. The most usual scenarios include:
  - $(a) \downarrow_i$ indicates that a is sent to child i (unicast);
  - $(a) \uparrow_i$ indicates that a is sent to parent i (unicast);
  - $(a) \downarrow_\forall$ indicates that a is sent to all children (broadcast);
  - $(a) \uparrow_\forall$ indicates that a is sent to all parents (broadcast);
  - $(a) \updownarrow_\forall$ indicates that a is sent to all neighbours (broadcast).

  All symbols sent via one *generic rule* to the same destination form one single *message* and they travel together as one single block (even if the generic rule has multiple instantiations).

Example. To explain our combined instantiation and rewriting mode, let us consider a cell, σ, containing three counter-like complex symbols, $c(1^2)$, $c(1^2)$, $c(1^3)$, and the four possible instantiation⊗rewriting modes of the following "decrementing" rule:

$$\boxed{(\rho_\alpha)\ S_1\ c(1X) \rightarrow_\alpha S_2\ c(X), \text{where } \alpha \in \{\texttt{min},\texttt{max}\} \times \{\texttt{min},\texttt{max}\}.}$$

1. If $\alpha = \texttt{min}\otimes\texttt{min}$, rule $\rho_{\texttt{min}\otimes\texttt{min}}$ nondeterministically generates and applies (in the `min` mode) *one* of the following two rule instances:

$$(\rho'_1)\quad S_1\ c(1^2) \rightarrow_{\texttt{min}} S_2\ c(1) \quad \text{or}$$
$$(\rho''_1)\quad S_1\ c(1^3) \rightarrow_{\texttt{min}} S_2\ c(1^2).$$

   Using (ρ'_1), cell σ ends with counters $c(1)$, $c(1^2)$, $c(1^3)$. Using (ρ''_1), cell σ ends with counters $c(1^2)$, $c(1^2)$, $c(1^2)$.

2. If $\alpha = \texttt{max}\otimes\texttt{min}$, rule $\rho_{\texttt{max}\otimes\texttt{min}}$ first generates and then applies (in the `min` mode) the following *two* rule instances:

$$(\rho'_2)\quad S_1\ c(1^2) \rightarrow_{\texttt{min}} S_2\ c(1) \quad \text{and}$$
$$(\rho''_2)\quad S_1\ c(1^3) \rightarrow_{\texttt{min}} S_2\ c(1^2).$$

   Using (ρ'_2) and (ρ''_2), cell σ ends with counters $c(1)$, $c(1^2)$, $c(1^2)$.

3. If $\alpha = \texttt{min}{\otimes}\texttt{max}$, rule $\rho_{\texttt{min}\otimes\texttt{max}}$ nondeterministically generates and applies (in the $\texttt{max}$ mode) *one* of the following rule instances:

$$\begin{aligned}&(\rho'_3) \quad S_1\ c(\mathit{1}^2) \rightarrow_{\texttt{max}} S_2\ c(\mathit{1}) \quad \text{or}\\ &(\rho''_3) \quad S_1\ c(\mathit{1}^3) \rightarrow_{\texttt{max}} S_2\ c(\mathit{1}^2).\end{aligned}$$

   Using (ρ'_3), cell σ ends with counters $c(\mathit{1})$, $c(\mathit{1})$, $c(\mathit{1}^3)$. Using (ρ''_3), cell σ ends with counters $c(\mathit{1}^2)$, $c(\mathit{1}^2)$, $c(\mathit{1}^2)$.

4. If $\alpha = \texttt{max}{\otimes}\texttt{max}$, rule $\rho_{\texttt{min}\otimes\texttt{max}}$ first generates and then applies (in the $\texttt{max}$ mode) the following *two* rule instances:

$$\begin{aligned}&(\rho'_4) \quad S_1\ c(\mathit{1}^2) \rightarrow_{\texttt{max}} S_2\ c(\mathit{1}) \quad \text{and}\\ &(\rho''_4) \quad S_1\ c(\mathit{1}^3) \rightarrow_{\texttt{max}} S_2\ c(\mathit{1}^2).\end{aligned}$$

   Using (ρ'_4) and (ρ''_4), cell σ ends with counters $c(\mathit{1})$, $c(\mathit{1})$, $c(\mathit{1}^2)$.

The interpretation of $\texttt{min}{\otimes}\texttt{min}$, $\texttt{min}{\otimes}\texttt{max}$ and $\texttt{max}{\otimes}\texttt{max}$ modes is straightforward. While other interpretations could be considered, the mode $\texttt{max}{\otimes}\texttt{min}$ indicates that the generic rule is instantiated as *many* times as possible, without *superfluous* instances (i.e. without duplicates or instances which are not applicable) and each one of the instantiated rules is applied *once*, if possible.

If a rule does not contain any non-ground term, then it has only one possible instantiation: itself. Thus, in this case, the instantiation is an *idempotent* transformation, and the modes $\texttt{min}{\otimes}\texttt{min}$, $\texttt{min}{\otimes}\texttt{max}$, $\texttt{max}{\otimes}\texttt{min}$, $\texttt{max}{\otimes}\texttt{max}$ fall back onto traditional modes $\texttt{min}$, $\texttt{max}$, $\texttt{min}$, $\texttt{max}$, respectively.

Special cases. Simple scenarios involving generic rules are sometimes semantically equivalent to loop-based sets of non-generic rules. For example, consider the rule

$$S_1\ a(x(I)\ y(J)) \ \rightarrow_{\texttt{max}\otimes\texttt{min}}\ S_2\ b(I)\ c(J),$$

where the cell's contents guarantee that I and J only match integers in ranges $[1, n]$ and $[1, m]$, respectively. Under these assumptions, this rule is equivalent to the following set of non-generic rules:

$$S_1\ a_{i,j}\ \rightarrow_{\texttt{min}} S_2\ b_i\ c_j,\ \forall i \in [1, n], j \in [1, m].$$

However, unification is a much more powerful concept, which cannot be generally reduced to simple loops.

Note. For all modes, the instantiations are *conceptually* created when rules are tested for applicability and are also *ephemeral*, i.e. they disappear at the end of the step. P system implementations are encouraged to directly apply high-level generic rules, if this is more efficient (it usually is); they may, but need not, start by transforming high-level rules into low-level rules, by way of instantiations.

Benefits. This type of generic rules allow (i) a reasonably fast parsing and processing of subcomponents, and (ii) algorithm descriptions with *fixed size alphabets* and *fixed sized rulesets*, independent of the size of the problem and number of cells in the system (often *impossible* with only atomic symbols).

A.3 Synchronous vs asynchronous

In our models, we do not make any *syntactic* difference between the synchronous and asynchronous scenarios; this is strictly a *runtime* assumption [4]. Any model is able to run on both the synchronous and asynchronous runtime "engines", albeit the results may differ.

In the *synchronous* scenario of traditional P systems, all rules in a step take together exactly *one* time unit and then all message exchanges (including loopback messages for in-symbols) are performed at the end of the step, in *zero* time (i.e. instantaneously). Alternatively, but logically equivalent, we may consider that all rules in a step are performed in *zero* time (i.e. instantaneously) and then all message exchanges are performed in exactly *one* time unit. We prefer the second interpretation, because it allows us to interpret synchronous runs as special cases of asynchronous runs.

In the *asynchronous* scenario, we still consider that rules in a step are performed in *zero* time (i.e. instantaneously), but then, to arrive at its destination, each message may (independently) take *any* finite real time in the $(0, 1]$ interval (i.e. travelling times are typically scaled to the travel time of the slowest message). Additionally, unless otherwise specified, we also assume that messages traveling on the same directed arc follow a *FIFO* rule, i.e. no fast message can overtake a slow progressing one. This definition closely emulates the standard definition used for asynchronous distributed algorithms [3]. Clearly, the asynchronous model is highly non-deterministic, but most useful algorithms manage to remain confluent.

In both scenarios, we need to cater for a particularity of P systems, where a cell may remain active after completing its current step and then will automatically start a new step, without necessarily receiving any new message. In contrast, in classical distributed models, nodes may only become active after receiving a new message, so there is no self-activation without messaging. We can solve this issue by (i) assuminging a hidden self-activation message that cells can post themselves at the end of the step (together with the *in-symbols*) and (ii) postulating that such self-addressed messages will arrive not later than any other messages coming from other cells.

Obviously, any algorithm that works correctly in the asynchronous mode will also work correctly in the synchronous mode, but the converse is *not* generally true: extra care may be needed to transform a correct synchronous algorithm into a correct asynchronous one; there are also general control layers, such as *synchronisers*, that can attempt to run a synchronous algorithm on an existing asynchronous runtime, but this does not always work [3].

Complexity measures. We consider a set of basic complexity measures similar to the ones used in the traditional *distributed algorithms* field.

- *Time complexity*: the supremum over all possible running times (which, although not perfect, is the most usual definition for the asynchronous time complexity).

– *Message complexity*: the number of exchanged messages.
– *Atomic complexity*: the number of atoms summed over all exchanged messages (analogous to the traditional bit complexity).

Other measure may be considered, such as various *static complexities*, *development time*, etc.

Tissue P Systems with Rule Production/Removal

Linqiang Pan[1,2], Bosheng Song[1, ⋆], and Gexiang Zhang[3]

[1] Key Laboratory of Image Information Processing and Intelligent Control of Education Ministry of China
School of Automation, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China
lqpan@mail.hust.edu.cn, boshengsong@hust.edu.cn
[2] School of Electric and Information Engineering
Zhengzhou University of Light Industry
Zhengzhou 450002, Henan, China
[3] Robotics Research Center and Key Laboratory of Fluid and Power Machinery
Xihua University, Chengdu 610039, China
zhgxdylan@126.com

Abstract. Tissue P systems are computational models inspired by the way of biochemical substance movement/exchange between two cells or between a cell and the environment, where all communication (symport/antiport) rules used in a system are initially set up and keep unchanged during any computation. In this work, a variant of tissue P systems, called tissue P systems with rule production/removal (abbreviated as TRPR P systems) is considered, where rules in a system are dynamically changed during a computation, that is, at any computation step new rules can be produced and some existing rules can be removed. The computation power of TRPR P systems is investigated. It is proved that Turing universality is achieved for TRPR P systems with one cell, and using symport rules of length at most 1, antiport rules of length at most 2 or symport rules of length at most 2 and working in a maximally parallel manner. We further show that TRPR P systems with two cells, using symport rules of length at most 1, and working in a flat maximally parallel manner, are Turing universal.

Keywords: Bio-inspired computing, Membrane computing, Tissue P system, Symport/antiport rule, Universality

1 Introduction

Cell is the basic unit of life, which can be viewed as an information processing device. Membrane computing is a computational paradigm inspired by the structure and functioning of living cells, which was initiated in 1998 by Gh. Păun [27] and the literature of this area has grown very fast in both theoretical and

⋆ Corresponding author.

practical aspects. Theoretical results include computation power [6, 9], computation complexity [42, 43], the variants of P systems [2, 47, 38], and the strategies of using rules in P systems [36, 37]. In the practical aspect, P systems are used to solve real application problems, such as fault diagnosis [32, 46], combinatorial optimization [48, 49], image processing [11, 13]). The computation devices in membrane computing are known as *P systems*, which have two main families: *cell-like P systems* [27], which have a hierarchical arrangement of membranes; and *tissue-like P systems* [21] or *neural-like P systems* [17], which have a net of cells or neurons. A comprehensive presentation of membrane computing can be found in [28, 30], and the most up-to-date source of information is available on the P systems webpage `http://ppage.psystems.eu`. The present work deals with tissue-like P systems, introduced in [20].

A tissue-like P system consists of cells that are described by a directed graph, where cells are nodes of a graph, and the environment is considered as a distinguished node, an arc between two nodes corresponds to a communication channel between two regions (two cells or a cell and the environment). If a communication channel between two regions exists, then objects in these two regions can communicate by means of communication (symport/antiport) rules [25]. Symport rules move objects between two regions in one direction, whereas antiport rules move objects between two regions in opposite directions.

Since the seminal definition of tissue P systems, several research lines have been developed [1, 7]. In [29], cell division was introduced into tissue P systems, and the `SAT` problem was solved in polynomial time by tissue P systems with cell division. In [10, 44], generalized communicating P systems were proposed, where only pairs of objects synchronously move across components. Tissue P systems with evolutional symport/antiport rules were proposed in [41], where objects can evolve when moving from one region to another region. In [4], energy associated with each cell is introduced in tissue P systems, and Turing universality is reached when maximally parallel mode or sequential mode enforced with priorities are considered. Tissue P systems with channel states controlling the communication between two cells or between a cell and the environment were proposed in [15], several Turing universality results are achieved, where the systems work in a maximally parallel way with sequential behavior on channels. Cell/symbol complexity of tissue P systems with symport/antiport rules was investigated in [5], it was proved that tissue P systems with two channels between the cell and the environment are Turing universal when having six cells and one symbol, or two cells and three symbols, or three cells and two symbols.

In standard tissue P systems and the variants mentioned above, all symport/antiport rules used in a tissue P system are initially set up and keep unchanged during any computation. Actually, living cells, objects in cells are moving in order to achieve particular functioning and chemical reactions can be affected by both the contents in cells and the environmental conditions. Thus, it is a rather natural idea to consider rule production or removal during the process of a computation in a tissue P system.

Creating new evolution rules during a computation has been considered in subsection 3.6.4 in [28], where a rule in a P system is used, we say the rule is "consumed", that is, we take into consideration the multiplicity of rules, to work with multisets of rules in the same way as we have worked with multisets of objects. Specifically, when a rule $r : u \to v; z$ is applied, a copy of r is consumed, and all rules indicated by z are created.

In [3], sequential P systems with regular control were proposed, where all rules are initially set up and divided into different subsets, and the application of subsets of rules is controlled by a regular language.

In this work, we consider *tissue P systems with rule production/removal* (abbreviated as TRPR P systems), where rules in a system are dynamically changed during a computation, that is, at any computation step new rules can be produced and some existing rules can be removed. With this regulation mechanism, the computation power of tissue P systems working in a maximally parallel manner and in a flat maximally parallel manner is investigated. Specifically, it is proved Turing universality is achieved for TRPR P systems with one cell, and using symport rules of length at most 1, antiport rules of length at most 2 or symport rules of length at most 2 and working in a maximally parallel manner. We further show that the result holds true also for TRPR P systems with two cells, using symport rules of length at most 1 and working in a flat maximally parallel manner.

2 Tissue P Systems with Rule Production/Removal

It is necessary to recall some basic concepts of formal language theory used in this work, for further details of formal language theory, one can refer to the monographs [33].

For an alphabet Γ (a finite non-empty set of symbols), we denote by Γ^* the set of all strings over Γ, and by $\Gamma^+ = \Gamma^* \setminus \{\lambda\}$ we denote the set of non-empty strings. The number of symbols in a string u is the *length* of the string, and it is denoted by $|u|$. The number of occurrences of symbol a in a string u is denoted by $|u|_a$.

A *multiset* over an alphabet Γ is a function m from Γ to the set $\mathbb{N}$ of natural numbers, which gives a nonnegative *multiplicity* $m(x)$ for each $x \in \Gamma$. Let m_1, m_2 be multisets over Γ. The union of m_1 and m_2, denoted by $m_1 + m_2$, is the multiset over Γ defined as $(m_1 + m_2)(x) = m_1(x) + m_2(x)$ for each $x \in \Gamma$. The relative complement of m_2 in m_1, denoted by $m_1 \setminus m_2$, is the multiset defined as $(m_1 \setminus m_2)(x) = m_1(x) - m_2(x)$ if $m_1(x) \geq m_2(x)$, and $(m_1 \setminus m_2)(x) = 0$ otherwise.

Next we introduce the definition of tissue P systems with rules production/removal.

Definition 1. *A tissue P system with rule production/removal (abbreviated as TRPR P systems) of degree $q \geq 1$ is a tuple $\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \mathcal{R}_2, i_{out})$, where*

- *Γ and $\mathcal{E}$ are finite alphabets such that $\mathcal{E} \subseteq \Gamma$;*
- *$\mathcal{M}_1, \ldots, \mathcal{M}_q$ are finite multisets over Γ;*
- *$\mathcal{R}_1$ is a finite set of rules of the following forms:*
  - *Symport rules: $(i, u/\lambda, j); \mathbf{r}$, for $0 \leq i \neq j \leq q, u \in \Gamma^+$;*
  - *Antiport rules: $(i, u/v, j); \mathbf{r}$, for $0 \leq i \neq j \leq q, u, v \in \Gamma^+$;*
- *$\mathcal{R}_2$ is a finite set of rules of the following forms:*
  - *Symport rules with rules production/removal: $(i, u/\lambda, j); \mathbf{r}$, for $0 \leq i \neq j \leq q, u \in \Gamma^+$, and $\mathbf{r}$ is a finite set whose elements are of the type $r \in \mathcal{R}_1$ or $-r$ with $r \in \mathcal{R}_1$;*
  - *Antiport rules with rules production/removal: $(i, u/v, j); \mathbf{r}$,for $0 \leq i \neq j \leq q, u, v \in \Gamma^+$, and $\mathbf{r}$ is a finite set whose elements are of the type $r \in \mathcal{R}_1$ or $-r$ with $r \in \mathcal{R}_1$;*
- *$i_{out} \in \{0, 1, \ldots, q\}$.*

A TRPR P system of degree $q \geq 1$ can be viewed as a set of q cells labelled by $1, \ldots, q$ such that: (a) $\mathcal{M}_1, \ldots, \mathcal{M}_q$ represent the finite multisets of objects initially placed in the q cells of the system; (b) $\mathcal{E}$ is the set of objects initially located in the environment of the system, all of them available in an arbitrary number of copies; (c) $\mathcal{R}_2$ is a finite set of rules initially present in the system; (d) i_{out} is the label of a distinguished region which will encode the output of the system. The term region i ($0 \leq i \leq q$) refers to cell i in case $1 \leq i \leq q$ and refers to the environment in case $i = 0$. The length of a symport rule with rules production/removal $(i, u/\lambda, j); \mathbf{r}$ (an antiport rule with rules production/removal $(i, u/v, j); \mathbf{r}$, respectively) is defined as $|u|$ ($|u| + |v|$, respectively).

Note that if set $\mathbf{r}$ contains both a creation rule r and a removal rule $-r$, then it is assumed that first a creation rule r is produced and then this rule r is removed immediately, this process takes one step. If set $\mathbf{r}$ contains a non-existing removal rule $-r$, then this removal rule $-r$ will not work in the following computation steps, that is, this removal rule $-r$ works only if there exists a rule r in the system.

A *configuration* of a TRPR P system at any instant is described by all multisets of objects over Γ associated with all cells in the system, and the multiset of objects over $\Gamma \backslash \mathcal{E}$ associated with the environment at that moment. Note that the objects from $\mathcal{E}$ have an arbitrary number of copies, hence they are not properly changed along the computation. The *initial configuration* is $(\mathcal{M}_1, \ldots, \mathcal{M}_q; \emptyset)$.

A symport rule with rules production/removal $(i, u/\lambda, j); \mathbf{r}$ is applicable to a configuration at a moment if there is a region i which contains multiset u. When such a rule is applied, the objects specified by u in region i are sent to region j, simultaneously, if $r_i \in \mathbf{r}$ then rule r_i is produced in the system; if $-r_i \in \mathbf{r}$ then rule r_i is removed from the system. An antiport rule with rules production/removal $(i, u/v, j); \mathbf{r}$ is applicable to a configuration at a moment if there is a region i which contains multiset u and a region j which contains multiset v. When such a rule is applied: (a) the objects specified by u in region i are sent to region j; (b) the objects specified by v in region j are sent to region i; and (c) if $r_i \in \mathbf{r}$ then rule r_i is produced in the system; if $-r_i \in \mathbf{r}$ then rule r_i is removed from the system.

The rules of a TRPR P system in this work are applied in two manners: (1) maximally parallel manner: at each step, we apply a multiset of rules which is maximal, no further rule can be added being applicable; (2) flat maximally parallel manner: in each step, in each cell, a maximal set of concurrently applicable rules is chosen and each rule in the set is applied exactly once.

Starting from the initial configuration and applying rules as described above, a sequence of consecutive configurations is obtained. Each passage from a configuration to a successor configuration is called a *transition*. A configuration is a *halting configuration* if no rule of the system is applicable to it. A sequence of transitions starting in the initial configuration is a *computation*. Only a computation reaching a halting configuration gives a result, encoded by the number of copies of objects present in the output region i_{out}.

We denote by $NOtP_m^{pr}(sym_{t_1}, anti_{t_2}, max)$ and $NOtP_m^{pr}(sym_{t_1}, anti_{t_2}, fmax)$ (resp., $NOtP_m(sym_{t_1}, anti_{t_2}, max)$) the family of sets of numbers computed by tissue P systems with at most m cells, and using symport rules with rules production/removal (resp., symport rules) of length at most t_1, antiport rules with rules production/removal (resp., antiport rules) of length at most t_2 working in a maximally parallel manner and in a flat maximally parallel manner. If one of the parameters m, t_1, t_2 is not bounded, then it is replaced with $*$.

The following fundamental result is known from Theorem 5.9 in Chapter 5 (R. Freund, A. Alhazov, Y. Rogozhin, S. Verlan, Communication P systems) in [30].

Theorem 1. *$NOtP_1(sym_1, anti_2, max) \cup NOtP_1(sym_2, max) \subseteq NFIN$ (the family of finite sets of non-negative integers).*

3 Universality of Tissue P Systems with Rules Production/Removal

A very useful characterization of NRE (the family of sets of numbers which are Turing computable) is obtained by means of *register machines*, we here introduce the notion of register machines.

A register machine is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is a set of labels, l_0 is the label of the initial instruction and l_h is the label of the halting instruction, and I is a set of instructions of the form $l_i : (op(i), l_j, l_k)$ such that $op(i)$ is an operation on register i of M, l_i, l_j, l_k are labels from I, $l_i \neq l_h$. Give an instruction $l_i : (op(i), l_j, l_k)$, if operation $op(i)$ can be applied to register i, then one continues with the instruction with label l_j, otherwise one continues with the instruction with label l_k.

The instructions $l_i : (op(i), l_j, l_k)$ are of the following forms:

– $l_i : (\mathtt{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k, non-deterministically chosen);
– $l_i : (\mathtt{SUB}(r), l_j, l_k)$ (if the contents of register r are greater than zero, then subtract 1 from register i, and go to the instruction with label l_j; otherwise,

do not change the contents of register i, and go to the instruction with label l_k);
- l_h : `HALT` (the halt instruction).

A register machine M recognizers the set $N(M)$ of all natural numbers such that M starts with the initial instruction l_0 and halts in halting instruction. It is known that register machines are equivalent to Turing machines in the sense that they recognize the same family of sets of numbers, hence they characterize NRE [22].

3.1 Tissue P Systems with Rules Production/Removal Working in the Maximally Parallel Manner

In this subsection, we prove that tissue P systems with one cell and using symport rules with rules production/removal of length at most 2 or using symport rules with rules production/removal of length at most 1, antiport rules with rules production/removal of length at most 2, and working in the maximally parallel manner can generate all recursively enumerable sets of numbers, i.e., they characterize NRE. However, in the case of standard tissue P systems with symport/antiport rules, these Turing universality results cannot be obtained by such P systems (see Theorem 1).

Theorem 2. $NOtP_1^{pr}(sym_1, anti_2, max) = NRE$.

Proof. We only need to prove the inclusion $NOtP_1^{pr}(sym_1, anti_2, max) \supseteq NRE$, and the reverse inclusion follows from the Church-Turing thesis.

We use the characterization of NRE by means of register machines. Let $M = (m, H, l_0, l_h, I)$ be a register machine, which generates the set of numbers $N(M)$. We construct the TRPR P system of degree 1 to simulate register machine M.

$$\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \mathcal{R}, 1),$$

where

- $\Gamma = \{a_i \mid 1 \leq i \leq m\} \cup \{b, l, l', l'', l''' \mid b, l \in H\}$,
- $\mathcal{E} = \{l', l'', l''' \mid l \in H\}$,
- $\mathcal{M}_1 = \{b \mid b \in H\} \cup \{l_0\}$,

and the set of symport/antiport rules with rules production/removal is as follows:

- For each ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ of M, the following rules are produced in cell 1:
  $r_{i,1} : (1, l_i/l_i', 0); r_{i,2}$,
  $r_{i,2} : (1, b_i/a_r, 0); -r_{i,2}, r_{i,3}$,
  $r_{i,3} : (1, \lambda/b_i, 0); r_{i,4}, r_{i,5}$,
  $r_{i,4} : (1, l_i'/l_j, 0); -r_{i,4}, r_{j,1}, -r_{h,1}$,
  $r_{i,5} : (1, l_i'/l_k, 0); -r_{i,5}, r_{k,1}, -r_{h,1}$.

An ADD instruction l_i is simulated in four steps. At step 1, rule $r_{i,1}$ is applied, object l_i in cell 1 is exchanged with object l_i' in the environment, simultaneously, rule $r_{i,2}$ is produced. At the next step, rule $r_{i,2}$ is used, one copy of object a_r is sent into cell 1, object b_i is sent to the environment and this object will be sent back into cell 1 by using the created rule $r_{i,3}$, moreover, rule $r_{i,2}$ is removed. At step 3, by using rule $r_{i,3}$, rules $r_{i,4}, r_{i,5}$ are produced. At the next step, rules $r_{i,4}$ and $r_{i,5}$ are used non-deterministically. By applying rule $r_{i,4}$ (resp., $r_{i,5}$), object l_i' in cell 1 is exchanged with object l_j (resp., l_k) in the environment; simultaneously, rule $r_{j,1}$ (resp., $r_{k,1}$) is produced, rules $r_{i,4}$ and $r_{h,1}$ (if appeared) (resp., $r_{i,5}$ and $r_{h,1}$ (if appeared)) are removed. Hence, one copy of object a_r is introduced into cell 1 (simulating that the number stored in register r is increased by one), the system starts to simulate an instruction with label l_j or l_k. So the instruction l_i of M is correctly simulated by Π.

– For each SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ of M, the following rules are produced in cell 1:
  $r_{i,1} : (1, l_i/l_i', 0); r_{i,2},$
  $r_{i,2} : (1, b_i/l_i'', 0); -r_{i,2}, r_{i,3}, r_{i,4},$
  $r_{i,3} : (1, a_r/b_i, 0); r_{i,5}, r_{i,6},$
  $r_{i,4} : (1, l_i'/l_i''', 0); -r_{i,4}, r_{i,7},$
  $r_{i,5} : (1, l_i''/\lambda, 0); -r_{i,5}, -r_{i,7},$
  $r_{i,6} : (1, l_i'''/l_j, 0); -r_{i,6}, r_{j,1}, -r_{h,1},$
  $r_{i,7} : (1, l_i'''/b_i, 0); -r_{i,7}, r_{i,8},$
  $r_{i,8} : (1, l_i''/l_k, 0); -r_{i,8}, r_{k,1}, -r_{h,1}.$

A SUB instruction l_i is simulated in the following way. At step 1, rule $r_{i,1}$ is used, object l_i' is sent into cell 1, simultaneously, rule $r_{i,2}$ is produced. At step 2, by using rule $r_{i,2}$, object b_i is sent to the environment, object l_i'' is sent into cell 1, simultaneously, rule $r_{i,2}$ is removed, and rules $r_{i,3}, r_{i,4}$ are produced. In what follows, there are two cases.

– There is at least one copy of object a_r in cell 1 (corresponding to that the number stored in register r is grater than 0). In this case, at step 3, rules $r_{i,3}$ and $r_{i,4}$ are enabled. By using rule $r_{i,3}$, object a_r in cell 1 is exchanged with object b_i in the environment, and rules $r_{i,5}, r_{i,6}$ are produced. By applying rule $r_{i,4}$, object l_i' in cell 1 is exchanged with l_i''' in the environment, simultaneously, rule $r_{i,4}$ is removed, and rule $r_{i,7}$ is produced. At the next step, rules $r_{i,5}$ and $r_{i,6}$ are enabled. By using rule $r_{i,5}$, object l_i'' is sent to the environment, and rules $r_{i,5}, r_{i,7}$ are removed. By applying rule $r_{i,6}$, object l_i''' in cell 1 is exchanged with object l_j in the environment, rules $r_{i,6}$ and $r_{h,1}$ (if appeared) are removed, and rule $r_{j,1}$ is produced. In this case, one copy of object a_r in cell 1 is consumed (simulating that the number stored in register r is decreased by one), and the system starts to simulate the instruction l_j.
– There is no object a_r in cell 1 (corresponding to that the number stored in register r is 0). In this case, at step 3, only rule $r_{i,4}$ can be used, object l_i''' is sent into cell 1, rule $r_{i,4}$ is removed, and rule $r_{i,7}$ is produced. At the next step, rule $r_{i,7}$ is enabled and applied, object l_i''' in cell 1 is exchanged with

object b_i in the environment, simultaneously, rule $r_{i,7}$ is removed, rule $r_{i,8}$ is produced. At step 5, by using rule $r_{i,8}$, object l_k is sent into cell 1, rules $r_{i,8}$ and $r_{h,1}$ (if appeared) are removed, and rule $r_{k,1}$ is produced. Hence, the system starts to simulate the instruction l_k.

Hence, the SUB instruction of M is correctly simulated by system Π.

When object l_h appears in cell 1, rule $r_{h,1}$ is produced, simultaneously, this rule is removed at the same step, no rule can be used in the system, and the computation halts. The number of the copies of object a_1 in cell 1 corresponds to the result of the computation, hence $N(M) = N(\Pi)$.

Theorem 3. $NOtP_1^{pr}(sym_2, max) = NRE$.

Proof. We only need to prove the inclusion $NOtP_1^{pr}(sym_2, max) \supseteq NRE$, and the reverse inclusion follows from the Church-Turing thesis.

We use the characterization of NRE by means of register machines. Let $M = (m, H, l_0, l_h, I)$ be a register machine, which generates the set of numbers $N(M)$. We construct the TRPR P system of degree 1 to simulate register machine M.

$$\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \mathcal{R}, 1),$$

where

– $\Gamma = \{a_i \mid 1 \leq i \leq m\} \cup \{b, b', l, l', l'', l''' \mid b, l \in H\}$,
– $\mathcal{E} = \{l', l'', l''' \mid l \in H\}$,
– $\mathcal{M}_1 = \{b, b' \mid b \in H\} \cup \{l_0\}$,

and the set of symport rules with rules production/removal is as follows:

– For each ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ of M, the following rules are produced in cell 1:
  $r_{i,1} : (1, b_i l_i/\lambda, 0); r_{i,2}$,
  $r_{i,2} : (1, \lambda/a_r b_i, 0); -r_{i,2}, r_{i,3}$,
  $r_{i,3} : (1, b_i/\lambda, 0); -r_{i,3}, r_{i,4}, r_{i,5}$,
  $r_{i,4} : (1, \lambda/b_i l_j, 0); -r_{i,4}, r_{j,1}, -r_{h,1}$,
  $r_{i,5} : (1, \lambda/b_i l_k, 0); -r_{i,5}, r_{k,1}, -r_{h,1}$.

An ADD instruction l_i is simulated in four steps. At step 1, rule $r_{i,1}$ is used, objects $b_i l_i$ in cell 1 are sent to the environment, and rule $r_{i,2}$ is produced. At the next step, rule $r_{i,2}$ is applied, objects $a_r b_i$ are sent into cell 1, simultaneously, rule $r_{i,2}$ is removed and rule $r_{i,3}$ is produced. At step 3, object b_i is sent to the environment, rule $r_{i,3}$ is removed, and rules $r_{i,4}$ and $r_{i,5}$ are produced. At step 4, rules $r_{i,4}$ and $r_{i,5}$ are used non-deterministically. By applying rule $r_{i,4}$ (resp., $r_{i,5}$), objects $b_i l_j$ (resp., $b_i l_k$) are sent into cell 1; simultaneously, rules $r_{i,4}$ and $r_{h,1}$ (if appeared) (resp., $r_{i,5}$ and $r_{h,1}$ (if appeared)) are removed, rule $r_{j,1}$ (resp., $r_{k,1}$) is produced. Hence, one copy of object a_r is introduced into cell 1 (simulating that the number stored in register r is increased by one), the system starts to simulate an instruction with label l_j or l_k. So the instruction l_i of M is correctly simulated by Π.

– For each SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ of M, the following rules are produced in cell 1:
$r_{i,1} : (1, b_i l_i/\lambda, 0); r_{i,2}$,
$r_{i,2} : (1, \lambda/b_i l_i', 0); -r_{i,2}, r_{i,3}, r_{i,4}$,
$r_{i,3} : (1, a_r b_i/\lambda, 0); -r_{i,3}, r_{i,6}, r_{i,8}$,
$r_{i,4} : (1, b_i' l_i'/\lambda, 0); r_{i,5}, r_{i,10}$,
$r_{i,5} : (1, \lambda/b_i' l_i'', 0)$,
$r_{i,6} : (1, \lambda/b_i l_i''', 0); r_{i,7}, -r_{i,10}$,
$r_{i,7} : (1, b_i' l_i'''/\lambda, 0); r_{i,9}$,
$r_{i,8} : (1, l_i''/\lambda, 0); -r_{i,8}$,
$r_{i,9} : (1, \lambda/b_i' l_j, 0); -r_{i,9}, r_{j,1}, -r_{h,1}$,
$r_{i,10} : (1, b_i l_i''/\lambda, 0); -r_{i,3}, r_{i,11}$,
$r_{i,11} : (1, \lambda/b_i l_k, 0); -r_{i,11}, r_{k,1}, -r_{h,1}$.

A SUB instruction l_i is simulated in the following way. At step 1, rule $r_{i,1}$ is used, objects $b_i l_i$ are sent to the environment, simultaneously, rule $r_{i,2}$ is produced. At step 2, by using rule $r_{i,2}$, objects $b_i l_i'$ are sent into cell 1, rule $r_{i,2}$ is removed, and rules $r_{i,3}, r_{i,4}$ are produced. In what follows, there are two cases.

– There is at least one copy of object a_r in cell 1 (corresponding to that the number stored in register r is grater than 0). In this case, at step 3, rules $r_{i,3}$ and $r_{i,4}$ are enabled. By using rule $r_{i,3}$, objects $a_r b_i$ in cell 1 are sent to the environment, rule $r_{i,3}$ is removed, and rules $r_{i,6}, r_{i,8}$ are produced. By applying rule $r_{i,4}$, objects $b_i' l_i'$ in cell 1 are sent to the environment, simultaneously, rules $r_{i,5}, r_{i,10}$ are produced. At the next step, rules $r_{i,5}$ and $r_{i,6}$ are enabled. By using rule $r_{i,5}$, objects $b_i' l_i''$ are sent into cell 1. By applying rule $r_{i,6}$, objects $b_i l_i'''$ are sent into cell 1, rule $r_{i,10}$ is removed, and rule $r_{i,7}$ is produced. At step 5, rules $r_{i,7}$ and $r_{i,8}$ are enabled. By using rule $r_{i,7}$, objects $b_i' l_i'''$ are sent to the environment, rule $r_{i,9}$ is produced. By applying rule $r_{i,8}$, object l_i'' is sent to the environment, and this rule is removed. At step 6, objects $b_i' l_j$ are sent into cell 1 by using rule $r_{i,9}$, where rule $r_{j,1}$ is produced, and rules $r_{i,9}$ and $r_{h,1}$ (if appeared) are removed. In this case, one copy of object a_r in cell 1 is consumed (simulating that the number stored in register r is decreased by one), and the system starts to simulate the instruction l_j.
– There is no object a_r in cell 1 (corresponding to that the number stored in register r is 0). In this case, at step 3, only rule $r_{i,4}$ can be used, objects $b_i' l_i'''$ are sent to the environment, rules $r_{i,5}$ and $r_{i,10}$ are produced. At the next step, rule $r_{i,5}$ is enabled and applied, objects $b_i' l_i''$ are sent into cell 1. At step 5, by using rule $r_{i,10}$, objects $b_i l_i''$ are sent to the environment, rule $r_{i,3}$ is removed, and rule $r_{i,11}$ is produced. At step 6, by using rule $r_{i,11}$, objects $b_i l_k$ are sent into cell 1, $r_{i,11}$ and $r_{h,1}$ (if appeared) are removed, and rule $r_{k,1}$ is produced. Hence, the system starts to simulate the instruction l_k.

Hence, the SUB instruction of M is correctly simulated by system Π.

When object l_h appears in cell 1, rule $r_{h,1}$ is produced, simultaneously, this rule is removed at the same step, no rule can be used in the system, and the computation halts. The number of the copies of object a_1 in cell 1 corresponds to the result of the computation, hence $N(M) = N(\Pi)$.

3.2 Tissue P Systems with Rules Production/Removal Working in the Flat Maximally Parallel Manner

Flat maximal parallelism of using rules was first considered in [16, 45] and then further investigated in [23, 39], where in each step, in each membrane, a maximal set of applicable rules is chosen and each rule in the set is applied exactly once. In this subsection, we prove that TRPR P system with two cells and using symport rules with rules production/removal of length at most 1 working in the flat maximally parallel manner can generate all recursively enumerable sets of numbers.

Theorem 4. $NOtP_2^{pr}(sym_1, fmax) = NRE$.

Proof. We only need to prove the inclusion $NOtP_1^{pr}(sym_1, fmax) \supseteq NRE$, and the reverse inclusion follows from the Church-Turing thesis.

We use the characterization of NRE by means of register machines. Let $M = (m, H, l_0, l_h, I)$ be a register machine, which generates the set of numbers $N(M)$. We construct the tissue P system of degree 1 to simulate register machine M.

$$\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \mathcal{M}_2, \mathcal{R}, 1),$$

where

– $\Gamma = \{a_i \mid 1 \leq i \leq m\} \cup \{b, l, l', l'' \mid b, l \in H\}$,
– $\mathcal{E} = \{l', l'' \mid l \in H\}$,
– $\mathcal{M}_1 = \{b \mid b \in H\} \cup \{l_0\}$, $\mathcal{M}_2 = \emptyset$,

and the set of producing/removing rules is as follows:

– For each ADD instruction l_i : $(\mathtt{ADD}(r), l_j, l_k)$ of M, the following rules are produced in cell 1 and in cell 2:
  $r_{i,1} : (1, l_i/\lambda, 0); r_{i,2}, r_{i,3}$,
  $r_{i,2} : (1, \lambda/a_r, 0); -r_{i,2}$,
  $r_{i,3} : (1, b_i/\lambda, 0); -r_{i,3}, r_{i,4}, r_{i,5}$,
  $r_{i,4} : (1, \lambda/b_i, 0); r_{i,6}$,
  $r_{i,5} : (2, \lambda/b_i, 0); r_{i,7}, r_{i,8}$,
  $r_{i,6} : (1, \lambda/l_j, 0); -r_{i,6}, r_{j,1}, -r_{h,1}$,
  $r_{i,7} : (1, \lambda/b_i, 2)$,
  $r_{i,8} : (1, \lambda/l_k, 0); -r_{i,8}, r_{k,1}, -r_{h,1}$.

An ADD instruction l_i is simulated in the following way. At step 1, rule $r_{i,1}$ is used, object l_i in cell 1 is sent to the environment, and rules $r_{i,2}, r_{i,3}$ are produced. At the next step, rules $r_{i,2}$ and $r_{i,3}$ are enabled. By using rule $r_{i,2}$, one copy of object a_r is sent into cell 1 (due to the flat maximal parallelism), simultaneously, rule $r_{i,2}$ is removed. At step 3, object b_i is sent to the environment, rule $r_{i,3}$ is removed, and rules $r_{i,4}$ and $r_{i,5}$ are produced. At step 4, rules $r_{i,4}$ and $r_{i,5}$ are used non-deterministically. By applying rule $r_{i,4}$ (resp., $r_{i,5}$), object b_i is sent into cell 1 (resp., cell 2); simultaneously, rule $r_{i,6}$ (resp., $r_{i,7}, r_{i,8}$) is produced. At step 4, by applying rule $r_{i,6}$, only one copy of object l_j is sent into cell 1 due to the flat maximal parallelism, simultaneously, rules $r_{i,6}$ and $r_{h,1}$ (if appeared) are removed, and rule $r_{j,1}$ is produced. By using rules $r_{i,7}, r_{i,8}$, object b_i in cell 2 is sent to cell 1, one copy of object l_k in the environment is sent into cell 1 due to the flat maximal parallelism, rules $r_{i,8}$ and $r_{h,1}$ (if appeared) are removed, and rule $r_{k,1}$ is produced. Hence, one copy of object a_r is introduced into cell 1 (simulating that the number stored in register r is increased by one), the system starts to simulate an instruction with label l_j or l_k. So the instruction l_i of M is correctly simulated by Π.

- For each SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ of M, the following rules are produced in cell 1:
  $r_{i,1} : (1, l_i/\lambda, 0); r_{i,2}, r_{i,3},$
  $r_{i,2} : (1, a_r/\lambda, 0); -r_{i,2}, r_{i,4},$
  $r_{i,3} : (1, b_i/\lambda, 0); -r_{i,2}, -r_{i,3}, r_{i,5}, r_{i,6},$
  $r_{i,4} : (1, \lambda/l'_i, 0); -r_{i,4}, r_{i,7},$
  $r_{i,5} : (1, \lambda/l''_i, 0), -r_{i,5}, r_{i,8},$
  $r_{i,6} : (1, \lambda/b_i, 0); -r_{i,6},$
  $r_{i,7} : (1, l'_i/\lambda, 0); -r_{i,7}, r_{i,9}, -r_{i,10},$
  $r_{i,8} : (1, l''_i/\lambda, 0); -r_{i,8}, r_{i,10},$
  $r_{i,9} : (1, \lambda/l_j, 0); -r_{i,9}, r_{j,1}, -r_{h,1},$
  $r_{i,10} : (1, \lambda/l_k, 0); -r_{i,10}, r_{k,1}, -r_{h,1}.$

A SUB instruction l_i is simulated in the following way. At step 1, rule $r_{i,1}$ is used, object l_i is sent to the environment, simultaneously, rules $r_{i,2}, r_{i,3}$ are produced. In what follows, there are two cases.

- There is at least one copy of object a_r in cell 1 (corresponding to that the number stored in register r is grater than 0). In this case, at step 2, rules $r_{i,2}$ and $r_{i,3}$ are enabled. By using rule $r_{i,2}$, object a_r in cell 1 is sent to the environment, rule $r_{i,2}$ is removed, and rule $r_{i,4}$ is produced. By applying rule $r_{i,3}$, object b_i in cell 1 is sent to the environment, simultaneously, rules $r_{i,2}, r_{i,3}$ are removed, rules $r_{i,5}, r_{i,6}$ are produced. At the next step, rules $r_{i,4}, r_{i,5}, r_{i,6}$ are enabled. By using rule $r_{i,4}$, one copy of object l'_i is sent into cell 1 due to the flat maximal parallelism, simultaneously, rule $r_{i,4}$ is removed and rule $r_{i,7}$ is produced. By applying rule $r_{i,5}$, one copy of object l''_i is sent into cell 1 (flat maximal parallelism), rule $r_{i,5}$ is removed, and rule $r_{i,8}$ is produced. By using rule $r_{i,6}$, object b_i is sent back to cell 1, and this rule is removed. At step 4, rules $r_{i,7}$ and $r_{i,8}$ are enabled. By using rule $r_{i,7}$, object l'_i is sent to the environment, rules $r_{i,7}, r_{i,10}$ are removed, rule $r_{i,9}$ is produced. By applying rule $r_{i,8}$, object l''_i is sent to the environment, and this rule is removed, rule $r_{i,10}$ is produced. Note that at step 4, rule $r_{i,10}$ is produced by using rule $r_{i,8}$ and this rule is removed by using rule $r_{i,7}$, hence after step 4, there is no rule $r_{i,10}$ in the system. At step 5, only rule $r_{i,9}$ is enabled and applied, only one copy of object l_j is sent into cell 1 due to the flat maximal parallelism, simultaneously, rule $r_{j,1}$ is produced, and rules $r_{i,9}$ and $r_{h,1}$ (if appeared) are removed. In this case, one copy of object a_r in cell 1 is consumed (simulating that the number stored in register r is decreased by one), and the system starts to simulate the instruction l_j.

– There is no object a_r in cell 1 (corresponding to that the number stored in register r is 0). In this case, at step 2, only rule $r_{i,3}$ can be used, objects b_i is sent to the environment, rules $r_{i,2}$ and $r_{i,3}$ are removed, rules $r_{i,5}$ and $r_{i,6}$ are produced. At the next step, rules $r_{i,5}$ and $r_{i,6}$ are enabled and applied, and rule $r_{i,8}$ is produced. At step 4, by using rule $r_{i,8}$, object l_i'' is sent to the environment, rule $r_{i,10}$ is produced. At step 5, by using rule $r_{i,10}$, only one copy of object l_k is sent into cell 1 due to the flat maximal parallelism, $r_{i,10}$ and $r_{h,1}$ (if appeared) are removed, and rule $r_{k,1}$ is produced. Hence, the system starts to simulate the instruction l_k.

Hence, the SUB instruction of M is correctly simulated by system Π.

When object l_h appears in cell 1, rule $r_{h,1}$ is produced, simultaneously, this rule is removed at the same step, no rule can be used in the system, and the computation halts. The number of the copies of object a_1 in cell 1 corresponds to the result of the computation, hence $N(M) = N(\Pi)$.

4 Conclusions and Discussions

In this work, tissue P systems with rules production/removal have been investigated. With the regulation mechanism of producing or removing rules, we have shown that Turing universality is achieved for tissue P systems with one cell, and using symport rules with rules production/removal of length at most 1, antiport rules with rules production/removal of length at most 2 or symport rules with rules production/removal of length at most 2 and working in a maximally parallel manner. Moreover, the result holds true also for tissue P systems with two cells, using symport rules with rules production/removal of length at most 1 and working in a flat maximally parallel manner.

Cell division [12, 14] or cell separation [24, 31], which can generate an exponential workspace in polynomial time, has been introduced into tissue P systems to solve **NP**-complete problems. It remains open how we construct tissue P systems with rules production/removal and cell division or cell separation to solve **NP**-complete problems with the condition that the number of initial rules is as small as possible.

Time-free manner of using rules was considered to solve **NP**-complete problems in membrane computing [34, 35, 40], where the correctness of the solution does not depend on the precise timing of the involved rules. It is of interest to construct tissue P systems with rules production/removal and cell division to solve **NP**-complete problems in a time-free manner.

Tissue P systems with cell division or with cell separation and without environment were considered in [8, 19], where the alphabet of the environment of such P systems is empty. It would be interesting to consider the computational efficiency of tissue P systems with rules production/removal and cell division or cell separation without environment.

Acknowledgements

The work was supported by National Natural Science Foundation of China (61602192, 61320106005 and 61033003), China Postdoctoral Science Foundation (2016M600592, 2017T100554), and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012).

References

1. Alhazov, A., Fernau, H., Freund, R., Ivanov, S., Siromoney, R., Subramanian, K.G.: Contextual Array Grammars with Matrix Control, Regular Control Languages, and Tissue P Systems Control. Theor. Comput. Sci. 682, 5–21 (2017).
2. Alhazov, A., Freund, R.: Variants of Small Universal P Systems with Catalysts. Fund. Informa. 138(1-2), 227–250 (2015).
3. Alhazov, A., Freund, R., Heikenwälder, H., Oswald, M., Rogozhin, Y., Verlan, S.: Sequential P Systems with Regular Control, In: Csuhaj-Varjú, E. (eds.) LNCS, vol. 7762, pp. 112–127. Springer, Heidelberg (2013).
4. Alhazov, A., Freund, R., Leporati, A., Oswald, M., Zandron, C.: (Tissue) P Systems with Unit Rules and Energy Assigned to Membranes. Fund. Informa. 74(4), 391–408 (2006).
5. Alhazov, A., Freund, R., Oswald, M.: Cell/Symbol Complexity of Tissue P Systems with Symport/Antiport. Int. J. Found. Comput. S. 17, 3–26 (2006).
6. Aman, B., Ciobanu, G.: Efficiently Solving The Bin Packing Problem Through Bio-Inspired Mobility. Acta Inform. 54(4), 435–445 (2017).
7. Besozzi, D., Busi, N., Cazzaniga, P., Ferretti, C., Leporati, A., Mauri, G., Pescini, D., Zandron, C.: (Tissue) P Systems with Cell Polarity. Math. Struct. Comput. Sci. 19(6), 1141–1160 (2009).
8. Christinal, H.A., Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Tissue-Like P Systems without Environment. Proceedings of the Eight Brainstorming Week on Membrane Computing, pp. 53–64. Sevilla, Spain, (2010).
9. Cienciala, L., Ciencialová, L.: Some New Results of P Colonies with Bounded Parameters. Nat. Comput. (2016) DOI: 10.1007/s11047-016-9591-0
10. Csuhaj-Varjú, E., Verlan, S.: On Generalized Communicating P Systems with Minimal Interaction Rules. Theor. Comput. Sci. 412, 124–135 (2011).
11. Díaz-Pernil, D., Berciano, A., Peña-Cantillana, F., Gutiérrez-Naranjo, M.A.: Segmenting Images with Gradient-Based Edge Detection Using Membrane Computing. Pattern Recogn. Let. 34(8), 846–855 (2013).
12. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A Uniform Family of Tissue P System with Cell Division Solving 3-Col in A Linear Time. Theor. Comput. Sci. 404, 76–87 (2008).
13. Díaz-Pernil, D., Peña-Cantillana, F., Gutiérrez-Naranjo, M.A.: A Parallel Algorithm for Skeletonizing Images by Using Spiking Neural P Systems. Neurocomputing 115, 81–91 (2013).

14. Díaz-Pernil, D., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Jiménez, Á.: Computational Efficiency of Cellular Division in Tissue-Like Membrane Systems. Rom. J. Inf. Sci. Tech. 11(3), 229–241 (2008).
15. Freund, R., Păun, Gh., Pérez-Jiménez, M.J.: Tissue P Systems with Channel States. Theor. Comput. Sci. 330, 101–116 (2005).
16. Freund, R., Verlan, S.: (Tissue) P Systems Working in the k-Restricted Minimally or Maximally Parallel Transition Mode. Nat. Comput. 10(2), 821–833 (2011).
17. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking Neural P Systems. Fund. Informa. 71(2-3), 279–308 (2006).
18. Krishna, S.N., Lakshmanan, K., Rama, R.: Tissue P Systems with Contextual and Rewriting Rules. In: Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) LNCS, vol. 2597, pp. 339–351. Springer, Heidelberg (2003).
19. Macías-Ramos, L.F., Pérez-Jiménez, M.J., Riscos-núñez, A., Rius-Font, M., Valencia-Cabrera, L.: The Efficiency of Tissue P Systems with Cell Separation Relies on the Environment. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) LNCS, vol. 7762, pp. 243–256. Springer, Heidelberg (2013).
20. Martín-Vide, C., Pazos, J., Păun, Gh., Rodríguez-Patón, A.: A New Class of Symbolic Abstract Neural Nets: Tissue P Systems. In: Ibarra, O.H., Zhang, L. (eds.) LNCS, vol. 2387, pp. 290–299. Springer, Heidelberg (2002).
21. Martín-Vide, C., Pazos, J., Păun, Gh., Rodríguez-Patón, A.: Tissue P Systems. Theor. Comput. Sci. 296(2), 295–326 (2003).
22. Minsky, M.L.: Computation: Finite and Infinite Machines, Prentice–Hall, Inc., Englewood Cliffs, New Jersey (1967).
23. Pan, L., Păun, Gh., Song, B.: Flat Maximal Parallelism in P Systems with Promoters. Theor. Comput. Sci. 623, 83–91 (2016).
24. Pan, L., Pérez-Jiménez, M.J.: Computational Complexity of Tissue-Like P Systems. J. Complexity 26(3), 296–315 (2010).
25. Păun, A., Păun, Gh.: The Power of Communication: P Systems with Symport/Antiport. New Generat. Comput. 20(3), 295–305 (2002).
26. Păun, A., Păun, Gh., Rozenberg, G.: Computing by Communication in Networks of Membranes. Int. J. Found. Comput. S. 13, 779–798 (2002).
27. Păun, Gh.: Computing with Membranes. J. Comput. Syst. Sci. 61(1), 108–143 (2000).
28. Păun, Gh.: Membrane Computing. An Introduction. Springer-Verlag, Berlin (2002).
29. Păun, Gh., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Tissue P Systems with Cell Division. Int. J. Comput. Commun. 3(3), 295–303 (2008).
30. Păun, Gh., Rozenberg, G., Salomaa, A.: (Eds.), The Oxford Handbook of Membrane Computing. Oxford University Press, New York (2010).
31. Pérez-Jiménez, M.J., Sosík, P.: An Optimal Frontier of the Efficiency of Tissue P Systems with Cell Separation. Fund. Informa. 138, 45–60 (2015).
32. Peng, H., Wang, J., Pérez-Jiménez, M.J., Wang, H., Shao, J., Wang, T.: Fuzzy Reasoning Spiking Neural P System for Fault Diagnosis. Inf. Sci. 235, 106–116 (2013).
33. Rozenberg, G., Salomaa, A.: (Eds.), Handbook of Formal Languages, 3 vols., Springer, Berlin (1997).
34. Song, T., Macías-Ramos, L.F., Pérez-Jiménez, M.J.: Time-Free Solution to SAT Problem Using P Systems with Active Membranes. Theor. Comput. Sci. 529, 61–68 (2014).

35. Song, B., Pan, L.: Computational Efficiency and Universality of Timed P Systems with Active Membranes. Theor. Comput. Sci. 567, 74–86 (2015).
36. Song, T., Pan, L.: Spiking Neural P Systems with Rules on Synapses Working in Maximum Spiking Strategy. IEEE T. Nanobiosci. 14, 465–477 (2015).
37. Song, T., Pan, L.: Spiking Neural P Systems with Rules on Synapses Working in Maximum Spikes Consumption Strategy. IEEE T. Nanobiosci. 14, 38–44 (2015).
38. Song, T., Pan, L.: Spiking Neural P Systems with Request Rules. Neurocomputing 193, 193–200 (2016).
39. Song, B., Pérez-Jiménez, M.J., Păun, Gh., Pan, L.: Tissue P Systems with Channel States Working in the Flat Maximally Parallel Way. IEEE Trans. NanoBiosci. 15(7), 645–656 (2016).
40. Song, B., Song, T., Pan, L.: A Time-free Uniform Solution to Subset Sum Problem by Tissue P Systems with Cell Division. Math. Struct. Comput. Sci. 27(1), 17–32 (2017).
41. Song, B., Zhang, C., Pan, L.: Tissue-Like P Systems with Evolutional Symport/Antiport Rules. Inf. Sci. 378, 177–193 (2017).
42. Sosík, P., Cienciala, L.: A Limitation of Cell Division in Tissue P Systems by PSPACE, J. Comput. Syst. Sci. 81, 473–484 (2015).
43. Sosík, P., Păun, A., Rodríguez-Patón, A.: P Systems with Proteins on Membranes Characterize PSPACE, Theor. Comput. Sci. 488, 78–95 (2013).
44. Verlan, S., Bernardini, F., Gheorghe, M., Margenstern, M.: Generalized Communicating P Systems, Theor. Comput. Sci. 404, 170–184 (2008).
45. Verlan, S., Quiros, J.: Fast Hardware Implementations of P Systems. In: Csuhaj-Varjú E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) LNCS, vol. 7762, pp. 404–423. Springer, Heidelberg (2013).
46. Wang, J., Shi, P., Peng, H., Pérez-Jiménez, M.J., Wang, T.: Weighted Fuzzy Spiking Neural P Systems. IEEE T. Fuzzy Syst. 21(2), 209–220 (2013).
47. Wu, T., Zhang, Z., Păun, Gh., Pan, L.: Cell-like Spiking Neural P Systems. Theor. Comput. Sci. 623, 180–189 (2016).
48. Zhang, G., Gheorghe, M., Pan, L., Pérez-Jiménez, M.J.: Evolutionary Membrane Computing: A Comprehensive Survey and New Results. Inf. Sci. 279, 528–551 (2014).
49. Zhang, G., Rong, H., Neri, F., Pérez-Jiménez, M.J.: An Optimization Spiking Neural P System for Approximately Solving Combinatorial Optimization Problems. Int. J. Neural Syst. 24(5), 1–16 (2014).

Reversing Steps in Membrane Systems Computations*

G. Michele Pinna

Dipartimento di Matematica e Informatica,
Università di Cagliari, Cagliari, Italy
gmpinna@unica.it

Abstract. The issue of reversibility in computational paradigms has gained interest in recent years. In this paper we investigate how to *reverse* steps in membrane systems computations. The problem is that computation steps in membrane systems do not preserve all the information that has to be used when reversing it. We try to formalize the relevant information needed, and we show that the proposed approach enjoy the so called *loop lemma*, which basically assures that the undoing obtained by reversely applying rules is correct.

1 Introduction

Membrane systems, introduced by G. Păun (see [22, 23] for a first account on membrane systems), are nowadays a popular and extensively studied computational paradigm inspired by how computations in the living cells take place.

The ingredients of this computational paradigm are a *membrane structure* (which is a tree-like structure), a multiset of *objects* associated to each membrane (spatial distribution of resources) and a set of *evolution rules* for each membrane (acting at local states). A computation step is performed by the application of a bunch of rules which consume objects from a membrane and produce objects in this membrane and possibly in the neighbouring membranes as well. All the possible instances of applicable rules are used, as it happens usually in nature, but this is not actually always needed to make the computational paradigm Turing equivalent (for instance, in [21] membrane systems where the rules have a special format are considered, and maximality is not required; similarly in [5] or [9] where, respectively, minimal parallelism or the presence of special objects called *catalysts* is considered). The computational paradigm has also been compared with many other paradigms (see [23] and the chapters therein for a fairly detailed account), and the relative expressivity has been studied (see, for instance, [6, 7] or [4]).

* Work partially supported by RAS, Sardinia Regional Government, Convenzione triennale tra la Fondazione di Sardegna e gli Atenei Sardi Regione Sardegna, (CUP: F72F16003030002), and P.I.A. 2013 "NOMAD"

Reversibility in computation paradigms is an issue that recently has received great attention[1]. Reversibility in nature has a quite precise meaning: once reached a certain state (say F) from a starting one (say I) with a sequence of steps, there is the capability of reaching again the state I, possibly applying the various steps in reverse order. Furthermore in nature also energy is considered, and the final balance after reversing steps should be zero. When focussing on computational devices, reversibility in general accounts on understanding how certain rules are applicable in reverse order and in particular the amount of information to be preserved. We do not discuss here further why reversibility is worth to be considered, and we refer to [16] for further motivations.

As already pointed out in [16], when reversibility is *backtracking* (the feature that certain *non deterministic* computations enjoy, which allows to *explore* all the possible alternatives), then reversibility is well understood. Just a suitable coding of the choices and of the applied rules is enough. It is much less clear in the case of *distributed* or *concurrent* systems, where the applications of the rules is done in a local fashion, and membrane systems have to be considered computing devices of this kind.

The aim of this paper is to investigate on how to *reverse* computation steps in membrane systems in such a way that if a configuration is reached from another one, there is a way to reverse this step.

Reversing computations can be achieved in various manners. One is to add suitable rules having the *reverse* effect of other rules, another approach is orthogonal to this one and it is based on devising on how to apply the same rule *reversely.* In the first way the fact that a computation is reversed is just a matter of observation on the results achieved by the computation itself, making the approach a sort of *simulation* of reversibility. Still some information about the computations may have to be considered (for instance, in the approach presented in [1] some information should be kept, as rules having as effect the dissolution of membranes are considered). In the case it is necessary to keep track of the previous existence of some membranes.

Another way focusses on how a rule is *reversely* applied to a given stage of the computation, and to apply a rule reversely it is often necessary to keep some information about the previous stages of the computation. The conditions to be established are such that a *loop lemma* can be proven. The loop lemma simply states that if one goes from a stage of the computation to another one by applying a bunch of rules together, then from the reached stage it is possible to *go back* by reversely applying the same bunch of rules. Though this seems to be an easy and minimal requirement, it is not obvious that it generally holds for concurrent and distributed systems. In fact, as discussed in [8] (see also [16]), further information have to be given in order to be able to reverse computations steps consistently. The added information have to guarantee that the previous stage of the computation can be *reconstructed* properly.

[1] This is testified by the series of workshops and conferences entitled *Reversible Computation* (RC) organized since 2009, which is a conference since 2013.

We achieve this result by enriching the notion of configuration of a membrane system with a *memory* which records the (minimal) information to be considered in reversing steps. The notion of memory we adopt is similar to the one of event structure associated to a membrane system developed in [2] and [20]. We are able to prove the loop lemma, though we get a weaker version which we will discuss. The choice of adding a memory is not the unique solution to the problems posed when reversing steps, as objects may be enriched with the full history. However the amount of information the memory may have is able to cover this other approach, hence we believe that what we propose is general enough.

Though reversibility often means to *fully* undo some steps, it is important to observe that this is not actually needed. In fact it seems more reasonable to allow to undo part of the steps rather than the whole one and this is feasible in our approach.

The paper in organized as follows. In the remaining part of this introduction we briefly recall how the issue of reversibility in membrane systems has been considered in literature. In the next section we give some background and in Section 3 we review the notion of membrane systems and formalize the notion of membrane systems computation. In Section 4 we first state in general what reversing computations in membrane systems may be, discussing briefly its limitation, and then develop our approach: in Subsection 4.1 we discuss how the information is added to configurations. Few ideas for future developments conclude the paper.

Reversibility in membrane systems: other approaches

Reversibility has been previously considered in membrane systems. In [1] O. Agrigoroaiei and G. Ciobanu present a first attempt to study reversibility in membrane systems. They develop a way to consider new *reversed* rules, called *dual* rules. Dual rules replace the original rules of the membrane system and reversed computations are studied with the aim at easing the search of appropriate solutions to problems *backward* rather than *forward* (and indeed the membrane systems introduced and studied in [1] are also called *dual* membrane systems). Thus computations are reversed as the whole system is actually reversed, which is different from undoing something.

Other papers consider when computations can be reversed, and they usually require that the membrane systems looked at are deterministic. For instance, [3] considers membrane systems where reversibility (or strong reversibility) means that every reachable configuration of the system can be *obtained* by a single configuration (and in the stronger version the reachability request is dropped), and the determinism issue is considered as well, meaning that every reachable configuration has just one successor configuration (again the strong version is the one requiring that the reachability request is dropped). In this paper conditions to achieve reversibility, strong reversibility, determinism and strong determinism are studied, and the expressivity of the associated systems is made precise. In [11] the problem of strong reversibility is further studied, and it is shown that it is decidable if a membrane system is strongly reversible. It is also worth to stress that the membrane systems considered have just one membrane.

In [18] the author considers membrane systems with *symport/antiport* rules, and it is shown that every reversible register machine can be simulated by a *deterministic* membrane system with *symport/antiport* rules. In [24] spiking neural systems are taken into account, and the investigations focus on the expressivity issue. Indeed it is shown that these systems are reversible because they are equivalent to reversible computing machines, and in all the above mentioned approaches the focus is on deterministic systems, whereas we consider reversibility without constraining it to special cases.

In [17] reversibility is considered as it is shown how to simulate Fredkin circuits with membrane systems, focussing on energy. Being Fredkin gates the base for achieving reversibility at circuit level, hence allowing to restore not only the state but also the energy, the fact that suitable membrane systems can simulate these circuits is quite relevant.

2 Background

We first fix some notation. With $\mathbb{N}$ we denote the set of *natural numbers* including zero, and with $\mathbb{N}^+$ the set of positive natural numbers. Given a set X, with $\mathbf{2}^X$ we indicate the set of subsets of X and with $\mathbf{2}^X_{\mathit{fin}}$ the set of *finite* subsets of X.

Given a set X, a partial order $\sqsubseteq$ on X is a reflexive, transitive and antisymmetric relation. Let $(X, \sqsubseteq)$ be a partially ordered set and $Y \subseteq X$, we say that Y has a *minimum* iff there exists $x \in X$ such that $\forall y \in Y$ it holds that $x \sqsubseteq y$. Dually it has a *maximum* iff there exists $x \in X$ such that $\forall y \in Y$ it holds that $y \sqsubseteq x$. The elements of $Y \subseteq X$ are referred to as *incomparable* iff $\forall y, y' \in Y.\ y \neq y'$ implies that $y \not\sqsubseteq y'$ and $y' \not\sqsubseteq y$. Given a partial order $(X, \sqsubseteq)$, with $max(X, \sqsubseteq)$ we denote the set of elements $Y \subseteq X$ such that (a) for each $y \in Y$ and for each $x \in X$ if $y \sqsubseteq x$ then $y = x$ (the element y is not dominated by any other element of X), and (b) for each $x \in X$ such that there is no $x' \in X$ with $x' \neq x$ and $x \sqsubseteq x'$, then $x \in Y$ (the set is the greatest subset of incomparable and maximal elements of X), and similarly with $min(X, \sqsubseteq)$ we denote the greatest subset of elements $Y \subseteq X$ that are minimal with respect to the partial order relation. Given two elements $x, y \in X$ such that $x \sqsubseteq y$, we say that x is an *immediate* predecessor of y iff $x \neq y$ and $\forall z \in X.\ x \sqsubseteq z \sqsubseteq y$ either implies $x = z$ or $z = y$. If x is the immediate predecessor of y, we indicate this with $x \mathrel{\hat{\sqsubseteq}} y$.

A partial order $(X, \sqsubseteq)$ is a *tree* if $\sqsubseteq$ is such that each subset $Y \subseteq X$ of incomparable elements has no maximum, and each subset $Y \subseteq X$ has a minimum. The minimum of X is called the *root* of the tree. We define some auxiliary partial functions over trees. Given a tree $(X, \sqsubseteq)$, we define the partial function $\mathsf{father} : X \to X$ by $\mathsf{father}(x) = y$ whenever $y \mathrel{\hat{\sqsubseteq}} x$. Clearly, the root of a tree has no father. The function $\mathsf{children} : X \to \mathbf{2}^X$ is defined by $\mathsf{children}(x) = \{y \in X \mid x \mathrel{\hat{\sqsubseteq}} y\}$. If x is a leaf, then $\mathsf{children}(x) = \emptyset$. We assume that the trees have a finite degree, namely for each node x we assume that $\mathsf{children}(x) \in \mathbf{2}^X_{\mathit{fin}}$.

Multisets. Given a set S, a *multiset* over S is a function $m : S \rightarrow \mathbb{N}$; we denote by ∂S the set of multisets of S. The *multiplicity* of an element s in m is given by $m(s)$. A multiset m over S is *finite* iff the set $\mathsf{dom}(m) = \{s \in S \mid m(s) \neq 0\}$ is finite and we always consider finite multisets. A multiset m such that $\mathsf{dom}(m) = \emptyset$ is called *empty*, and it is denoted by $\mathbf{0}$. The cardinality of a multiset is defined as $\#(m) = \sum_{s \in S} m(s)$. Given a multiset in ∂S and a subset $S' \subseteq S$, by $m|_{S'}$ we denote the multiset over S' such that $m|_{S'}(s) = m(s)$. We write $m \subseteq m'$ if $m(s) \leq m'(s)$ for all $s \in S$, and $m \subset m'$ if $m \subseteq m'$ and $m \neq m'$. The operator $\oplus$ denotes *multiset union*: $(m \oplus m')(s) = m(s) + m'(s)$. The operator $\ominus$ denotes *multiset difference*: $(m \ominus m')(s) =$ if $m(s) > m'(s)$ then $m(s) - m'(s)$ else 0. The *scalar product* of a number j with a multiset m is $(j \cdot m)(s) = j \cdot (m(s))$. We sometimes write a multiset $m \in \partial S$ as the sum $\oplus_{s \in S} m(s) \cdot s$, where we omit the summands whenever $m(s)$ is equal to 0. Finally we assume that all the operations defined so far extend (with overloading of notation) to vectors of multisets, applying the operations component-wise.

Membranes structure. The language of *membrane structure*, which we will denote with MS, is a language over the alphabet $\{[,]\}$, and it is defined inductively as follows:

- $[\,] \in \mathsf{MS}$, and
- if $\mu_1, \ldots, \mu_n \in \mathsf{MS}$ then also $[\mu_1 \ldots \mu_n] \in \mathsf{MS}$.

Two words in MS are *equivalent* whenever they represent the same tree up to isomorphisms, and a *membrane* μ is the equivalence class of all the words with respect to this equivalence. Observe that, given a membrane μ, a *matching pair* of parentheses is any substring of μ which is again a membrane. The number of membranes appearing in a membrane μ is calculated as follows:

$$\#_{\mathsf{MS}}(\mu) = \begin{cases} 1 & \text{if } \mu = [\,] \\ 1 + \sum_{i=1}^{k} \#_{\mathsf{MS}}(\mu_i) & \text{if } \mu = [\mu_1 \ldots \mu_k] \end{cases}$$

and to each membrane μ' appearing in a membrane μ, including μ itself, it is possible to associate an unique index i ranging from 1 to $\#_{\mathsf{MS}}(\mu)$, and we denote this index with $\mathsf{index}(\mu')$. If $\mu_i = [\mu_{i_1} \ldots \mu_{i_k}]$ then $\mathsf{father}(i_j) = i$ for $1 \leq j \leq k$, and $\mathsf{children}(i) = \{i_1, \ldots, i_k\}$. We assume that the index 1 is given to the root. Obviously the set $(\{1, \ldots, \#_{\mathsf{MS}}(\mu)\}, \sqsubseteq^*)$ is a tree, where $\mathsf{index}(\mu') \sqsubset \mathsf{index}(\mu_i)$ whenever $\mu' = [\mu_1 \ldots \mu_k]$, with $1 \leq i \leq k$, and $\sqsubseteq^*$ is the reflexive and transitive closure of $\sqsubset$,

3 Membrane Systems

We are now ready to recall the notion of *membrane system*. The main ingredients of a membrane system are three: a membrane structure, a multiset of objects associated to each membrane and a set of *evolution rules* associated to each membrane. The membrane structure represents the various compartments where the computations take place (in general simultaneously), and the conditions under which certain evolution rules can be applied is checked *locally*, *i.e.* in the same membrane to which the rules are associated. The result of the application of a rule has a more global effect, as it will be clear in the following.

We first fix a finite alphabet of (names of) objects (sometimes called molecules), that we denote with $\mathcal{O}$ and we fix an alphabet of *rule names*, that will be denoted with Name, and it will be ranged over by $\mathfrak{n}$.

Definition 1. *A* membrane system *over a set of objects* $\mathcal{O}$ *is a construct* $\Pi = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ *where:*

- μ *is a* membrane structure *with* n *membranes indexed from* 1 *to* n*, where* $n = \#_{\mathsf{MS}}(\mu)$*,*
- *each* w_i^0 *is a multiset over* $\mathcal{O}$ *associated with membrane* i*, and*
- *each* R_i *is a finite set of* reaction (or evolution) rules r *associated with the membrane* i*, each rule having the format* $r : u \to v$*, where* u *is a non empty finite multiset in* $\partial\mathcal{O}$*,* v *is a finite multiset over* $\mathcal{O} \times (\{\mathsf{here}, \mathsf{out}\} \cup \{\mathsf{in}_j \mid \mathsf{father}(j) = i\})$*, and* $\mathsf{name}(r) \in \mathsf{Name}$ *is the* name *of the rule* r*.*

The definition is almost standard, the difference is that we omitted the output membrane which is usually considered when one wishes to focus on what is calculated by a membrane system, and we focus on a rule format where a multiset of objects of a membrane are possibly *transformed* in multisets of objects in the same membrane and in the neighbouring ones (*i.e.* the father and the children). Two rules r, r' belonging to different sets of reaction rules (thus associated to different membranes) may be equal, where equal means that if $r : u \to v$ and $r' : u' \to v'$ then $u = u'$ and $v = v'$. We however assume that all the rules in a membrane system have distinct names, *i.e.* for each $r, r' \in \bigcup_{1 \leq i \leq n} R_i$, if $r \neq r'$ then $\mathsf{name}(r) \neq \mathsf{name}(r')$ and if $r = r'$ then there exists $k, j \in \{1, \ldots, n\}$ such that $r \in R_k$, $r' \in R_j$, $k \neq j$ and $\mathsf{name}(r) \neq \mathsf{name}(r')$. Given a rule $r \in \bigcup_{1 \leq i \leq n} R_i$, with $\mathsf{index}(r)$ we denote the index of the membrane this rule is associated to, thus if $r \in R_i$ then $\mathsf{index}(r) = i$.

The application of a rule $r : u \to v$ in a membrane i will *consume* the multiset u that must be in the membrane i and may cause the *production* of multisets not only in the same membrane i but also in the neighbouring membranes, if there are, namely those that are children of i and the $\mathsf{father}(i)$ membrane, if this exists. With $\mathcal{I}(r)$ we denote the set with the indices of the membranes where a rule r actually produces an object. Given a rule r, u is the *left hand side* of r and v is the *right hand side* of r, and they are denoted with $\mathsf{lhs}(r)$ and $\mathsf{rhs}(r)$, respectively. To simplify the notation, given a multiset z over $\mathcal{O} \times (\{\mathsf{here}, \mathsf{out}\} \cup \{\mathsf{in}_j \mid \mathsf{father}(j) = i\})$, with $z|_\alpha$ we denote the multiset on $\mathcal{O}$ obtained from z by considering all the elements with the second component equal to α, where $\alpha \in \{\mathsf{here}, \mathsf{out}, \mathsf{in}_1, \ldots, \mathsf{in}_n\}$. Given a rule r, its $\mathsf{rhs}(r) = v$ may be represented as $(v|_{\mathsf{here}}, \mathsf{here}) \oplus (v|_{\mathsf{out}}, \mathsf{out}) \oplus (v|_{\mathsf{in}_{j_1}}, \mathsf{in}_{j_1}) \oplus \ldots \oplus (v|_{\mathsf{in}_{j_k}}, \mathsf{in}_{j_k})$ where $\{j_1, \ldots, j_k\} = \mathsf{children}(\mathsf{index}(r))$. Observe that it may be that some of the $v|_\alpha$ are equal to $\mathbf{0}$. Given a rule r, the indices involved in the effect of this rule are $\{\mathsf{index}(r) \mid \mathsf{rhs}(r)|_{\mathsf{here}} \neq \mathbf{0}\} \cup \{\mathsf{father}(\mathsf{index}(r)) \mid \mathsf{rhs}(r)|_{\mathsf{out}} \neq \mathbf{0}\} \cup \{i \mid$

$i \in \mathsf{children}(\mathsf{index}(r)) \ \wedge \ \mathsf{rhs}(r)|_{\mathsf{in}_i} \neq \mathbf{0}\}$. We assume that, for each rule r, it holds that $\mathcal{I}(r) \neq \emptyset$, hence each rule has an effect different from the *annihilation* of all the objects involved[2].

Membrane Systems Evolution. A membrane system Π evolves from a configuration to another configuration as a consequence of the application of (multisets of) rules in each region. The rules are applied *simultaneously.* We start formalizing the notion of configuration of a membrane system.

Definition 2. *Let* $\Pi = (\mathcal{O}, \mu, w_1^0, \dots, w_n^0, R_1, \dots, R_n)$ *be a membrane system, then a* configuration *is a tuple* $C = (w_1, \dots, w_n)$ *where each* w_i *is a multiset over* $\mathcal{O}$. $C_0 = (w_1^0, \dots, w_n^0)$ *is the* initial *configuration of* Π.

A computation step of a membrane system is *triggered* by the application of multisets of rules in each membrane. These multisets of rules are collected in a vector.

Definition 3. *Let* $\Pi = (\mathcal{O}, \mu, w_1^0, \dots, w_n^0, R_1, \dots, R_n)$ *be a membrane system, then a* multi-rule vector $\overrightarrow{R}$ *is the tuple* $(\widehat{R}_1, \dots, \widehat{R}_n)$, *where* $\widehat{R}_i$ *is a multiset over* R_i.

The multi-rule vector $\overrightarrow{R}$ contains all the rules that have to be applied simultaneously to a configuration of a membrane system, with their proper multiplicities.

A multi-rule vector $\overrightarrow{R}$ is *enabled* at a configuration C whenever each multiset of objects in each region is greater than or equal to what all the rules to be applied in that region consume. Given a multi-rule vector $\overrightarrow{R}$, for each i between 1 and n we denote with $\mathsf{Lhs}(\overrightarrow{R})_i$ the multiset over $\mathcal{O}$ defined as follows: $\bigoplus_{r \in R_i} \widehat{R}_i(r) \cdot \mathsf{lhs}(r)$. The tuple of these multisets is denoted with $\mathsf{Lhs}(\overrightarrow{R})$.

Definition 4. *Let* $\Pi = (\mathcal{O}, \mu, w_1^0, \dots, w_n^0, R_1, \dots, R_n)$ *be a membrane system,* $\overrightarrow{R}$ *a multi-rule vector and* C *a configuration. Then* $\overrightarrow{R}$ *is* enabled *at* $C = (w_1, \dots, w_n)$ *if* $\forall i \in \{1, \dots, n\}$. $\mathsf{Lhs}(\overrightarrow{R})_i \subseteq w_i$. *We denote the enabling of a multi-rule vector* $\overrightarrow{R}$ *at a configuration* C *with* $C\,[\overrightarrow{R}\rangle$.

The effects of the *application* of a multi-rule vector $\overrightarrow{R}$ (which acts in all membranes concurrently) in the membrane i are the following: $\bigoplus_{r \in R_i} \widehat{R}_i(r) \cdot \mathsf{rhs}(r)|_{\mathsf{here}}$ is the effect of the rules in the same membrane, $(\bigoplus_{r \in R_{\mathsf{father}(i)}} \widehat{R}_{\mathsf{father}(i)}(r) \cdot \mathsf{rhs}(r)|_{\mathsf{in}_i})$ those of the rules in the father membrane, and finally $(\bigoplus_{j \in \mathsf{children}(i)} (\bigoplus_{r \in R_j} \widehat{R}_i(r) \cdot \mathsf{rhs}(r)|_{\mathsf{out}}))$ those from the children membranes. Like previously, these three parts are combined by using $\oplus$. For each membrane, we denote the effects by $\mathsf{Rhs}(\overrightarrow{R})_i$. The tuple of these effects is written as $\mathsf{Rhs}(\overrightarrow{R})$.

The following definition captures the notion of evolution of a membrane system with the application of a multi-rule vector $\overrightarrow{R}$.

[2] This requirement is reasonable when one imagine that reversing means undoing the effects of a rule, thus if a rule just serves to annihilate all the objects to be rewritten then one can imagine that such a rule can be always reversed, in any multiplicity.

Definition 5. *Let $\Pi = (\mathcal{O}, \mu, w_1^0, \dots, w_n^0, R_1, \dots, R_n)$ be a membrane system, $C = (w_1, \dots, w_n)$ be a configuration and $\overrightarrow{R} = (\widehat{R}_1, \dots, \widehat{R}_n)$ be a multi-rule vector such that $C\,[\overrightarrow{R}\rangle$. Then $\overrightarrow{R}$ can be* executed *and its execution leads to a configuration $C' = (w_1', \dots, w_n')$ where $w_i' = w_i \ominus \mathsf{Lhs}(\overrightarrow{R})_i \oplus \mathsf{Rhs}(\overrightarrow{R})_i$. The execution of a multi-rule vector $\overrightarrow{R}$ at a configuration C is denoted with $C\,[\overrightarrow{R}\rangle\, C'$.*

For the enabling and the execution of a multi-rule vector we adopt a notation resembling the one usually adopted for Petri nets, also because of the tight connections among these two formalisms (see [13, 12, 20, 6, 7] among others, or the chapter in [23]). Sometimes we will call an evolution step of a membrane system as a *reaction step*.

We now formalize the *chain* of "reactions" for a given membrane system: C_0 is a reaction sequence, and if $C_0\,[\overrightarrow{R}_1\rangle\, C_1 \dots C_{n-1}\,[\overrightarrow{R}_n\rangle\, C_n$ is a reaction sequence, and $C_n\,[\overrightarrow{R}\rangle\, C$, then $C_0\,[\overrightarrow{R}_1\rangle\, C_1 \dots C_n\,[\overrightarrow{R}\rangle\, C$ is also a reaction sequence. A configuration C is said to be *reachable* if there is a reaction sequence starting from the initial configuration and leading to C, *i.e.* $C_0\,[\overrightarrow{R}_1\rangle\, C_1 \dots C_{n-1}\,[\overrightarrow{R}_n\rangle\, C_n$ with $C = C_n$.

The evolution of membrane systems may have several strategies, and usually it is assumed that in each membrane all the applicable rules are actually applied in a *maximally parallel* way. Thus if $\overrightarrow{R}$ is enabled at the configuration C ($C\,[\overrightarrow{R}\rangle$) it is implicitly assumed that there is no rule r in any of the rules sets R_i such that $C\,[\overrightarrow{R}'\rangle$ where $\overrightarrow{R}'$ is obtained from $\overrightarrow{R}$ adding an instance of the rule r to the proper multiset. However, other strategies may be used, for instance maximality with respect to a specific membrane index (no rule associated to that membrane can be added to the multi-rule vector), or the rules to be applied are those involving the presence of a specific object called catalyst, or to each rule a readiness index can be associated and the criteria is to maximize the sum of these indices, or simply a priority can be attached to each rule and those enabled with highest priorities have to be applied. The various strategies that can be adopted have an influence on the expressiveness of the paradigm, that is not our concern, as we already mentioned in the introduction.

4 Reversing membrane system computations

Reversibility in membrane systems is strongly connected to the idea that computations are *deterministic*. Here we consider an approach which is more similar to the one taken when reversibility is considered in the realm of distributed and concurrent computations.

Rather than introducing new rules (reversed, like in dual membrane systems where the effect of *undoing* is obtained applying reversed rules) we formalize what the reverse application of a multi-rule vector is.

Definition 6. *Let $\Pi = (\mathcal{O}, \mu, w_1^0, \dots, w_n^0, R_1, \dots, R_n)$ be a membrane system, $C = (w_1, \dots, w_n)$ a configuration and $\overrightarrow{R}$ be a multi-rule vector. Then $\overrightarrow{R}$ is* reversely enabled *at C whenever, for all $i \in \{1, \dots, n\}$, it holds that $\mathsf{Rhs}(\overrightarrow{R})_i \subseteq w_i$, and it is denoted with $C\,\langle\overrightarrow{R}]$.*

The intuition is almost trivial: the enabling is done by checking on the effects of the application of rules. Observe that this fits easily when rule formats like symport/antiport are considered, or like in [7] where a more general format for rules is considered.

Definition 7. *Let $\Pi = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system, $C = (w_1, \ldots, w_n)$ a configuration and $\overrightarrow{R}$ be a multi-rule vector such that $C \langle \overrightarrow{R}]$. Then $\overrightarrow{R}$ can be* reversed *and the effects of reversing this multi-rule vector are, for all $i \in \{1, \ldots, n\}$, $w_i' = w_i \ominus \mathsf{Rhs}(\overrightarrow{R})_i \oplus \mathsf{Lhs}(\overrightarrow{R})_i$. We write $C \langle \overrightarrow{R}] \, C'$ to state that the configuration C' is the effect of reversing the multi-rule vector $\overrightarrow{R}$. In this case we say that $\overrightarrow{R}$ is* reversely executed.

Once we have established what reversely enabling and reverse execution might be, we start to connect these notion with the usual *forward* executions.

Proposition 1. *Let $\Pi = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system, $C = (w_1, \ldots, w_n)$ a configuration and $\overrightarrow{R}$ be a multi-rule vector such that $C [\overrightarrow{R}\rangle$, and let C' be the configuration reached executing $\overrightarrow{R}$,* i.e. *$C [\overrightarrow{R}\rangle \, C'$. Then $C' \langle \overrightarrow{R}]$.*

The loop lemma can be easily proven also in this setting:

Lemma 1 (Loop lemma). *Let $\Pi = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system, $C = (w_1, \ldots, w_n)$ a configuration and $\overrightarrow{R}$ be a multi-rule vector such that $C [\overrightarrow{R}\rangle$, and let C' be the configuration reached executing $\overrightarrow{R}$,* i.e. *$C [\overrightarrow{R}\rangle \, C'$. Then $C' \langle \overrightarrow{R}] \, C$.*

The proof of the following theorem is obvious.

Theorem 1. *Let Π be a membrane system, C a configuration and $\overrightarrow{R}$ be a multi-rule vector such that $C [\overrightarrow{R}\rangle C'$. Then there exists a multi-rule vector $\overrightarrow{R'}$ such that $C' \langle \overrightarrow{R'}] \, C$.*

Observe that not necessarily $\overrightarrow{R'}$ should be equal to $\overrightarrow{R}$. In fact they may differ.

Example 1. Consider the membrane system with just one membrane, the unique rule associated to the membrane are $r^1 = a \to (b, \mathsf{here})$ and $r^2 = a \oplus b \to (2b, \mathsf{here})$, and the initial configuration is $a \oplus b$. The rule r^1 is enabled at the initial configuration and its application leads to the configuration $2b$. Now also r^2 can be reversely applied at this configuration and the initial configuration can be obtained again.

The main problem is that membrane systems do not keep any information about the past, thus at a certain configuration it could be that a multi-rule vector $\overrightarrow{R}$ can be reversely executed even when no $\overrightarrow{R}'$ such that $\overrightarrow{R} \subseteq \overrightarrow{R}'$ has been "forwardly" executed. This contrasts the idea that reversibility is like undoing something that has been done previously.

Example 2. Consider the membrane system with 2 membranes $[\ [\ ]_2\ ]_1$, where the indices are the ones associated to the membranes, and with the following sets of rules: $\{r_1^1 : 2a \to (a \oplus b, \mathsf{here}) \oplus (b, \mathsf{in}_2), r_2^1 : b \to (a, \mathsf{here}) \oplus (c, \mathsf{in}_2), r_3^1 : a \oplus b \to (2a, \mathsf{here}) \oplus (b, \mathsf{in}_2), r_4^1 : a \to (b, \mathsf{here}) \oplus (c, \mathsf{in}_2), r_5^1 : 2a \to (b \oplus c, \mathsf{in}_2)\}$ are the rules associated to the first membrane, and $\{r_1^2 : b \to (2a, \mathsf{out}), r_2^2 : c \to (b, \mathsf{out}), r_3^2 : b \to (a, \mathsf{out}), r_4^2 : c \to (c, \mathsf{out})\}$ are those associated to the second membrane. The initial configuration is (w_1^0, w_2^0) where $w_1^0 = 2a \oplus b$ and $w_2^0 = \mathbf{0}$. The configuration $(2a \oplus b, b \oplus c)$ can be reached either executing the multi-rule vector $(r_1^1 \oplus r_2^1, \mathbf{0})$ or the one $(r_3^1 \oplus r_4^1, \mathbf{0})$. At this configuration these two multi-rule vectors are reversely enabled, but also the multi-rule vector $(r_5^1, \mathbf{0})$, and reversely executing it we would obtain the configuration $(2a, \mathbf{0})$ which is not reachable using the rules in the membrane system.

A similar problem is present in all the algebraic process calculi for which reversibility has been studied (see [8, 14, 10, 15] among others). The solution is usually to add a *memory* which helps to keep track of the evolution of the processes. Here we pursue a similar idea by adding information to configurations (membranes). We assume that Name contains $\perp$ as a name which is not associated to any rule.

4.1 Membranes with memory

Objects of a membrane system may be enriched by adding the name of the rule producing them. Thus objects would be $\mathcal{O} \times \mathsf{Name}$, and reversing a step would be to find out whether there are enough objects with specific rules names. The *forward* enabling would ignore the information on which rule produced the object, and the execution of the step would simply add the proper name of each object produced. This solution allow to undo just one step, as the information on the name of the rule of the *consumed* object are lost.

To be able to undo more steps we have to figure out a different structure, which we call memory and we will add it to configurations.

We briefly discuss what the *memory* in this case could be. The idea is rather simple: the memory is a labeled partial order, where the labeling gives a triple composed by an object, the index of a membrane and a rules name, thus $\langle o, i, \mathfrak{n} \rangle$ conveys the idea that the object o has been produced in the membrane i using the rule $\mathfrak{n}$.

Definition 8. *Let* Name *be a set of rules names such that* $\perp \in \mathsf{Name}$*, let* $\mathcal{O}$ *be a set of objects and let* $\mathcal{I}$ *be a set of indices. Then a* memory m *is the labeled partial order* $(X, \preceq, l)$ *where* $(X, \preceq)$ *is a partial order and* $l\colon X \to \mathcal{O} \times \mathcal{I} \times \mathsf{Name}$ *is a labeling mapping With* Mem *we denote the set of memories.*

Given an element of $(o, i, \mathfrak{n}) \in \mathcal{O} \times \mathcal{I} \times \mathsf{Name}$, we define some obvious projections operators, that carry over on multistes of $\mathcal{O} \times \mathcal{I} \times \mathsf{Name}$. $\mathsf{obj}_m\colon \mathcal{O} \times \mathcal{I} \times \mathsf{Name} \to \mathcal{O}$ is defined as $\mathsf{obj}_m(o, i, \mathfrak{n}) = o$, $\mathsf{i}_m\colon \mathcal{O} \times \mathcal{I} \times \mathsf{Name} \to \mathcal{I}$ as $\mathsf{i}_m(o, i, \mathfrak{n}) = i$, and finally $\mathsf{rule}_m\colon \mathcal{O} \times \mathcal{I} \times \mathsf{Name} \to \mathsf{Name}$ as $\mathsf{rule}_m(o, i, \mathfrak{n}) = \mathfrak{n}$. Given $\mathsf{m} = (X, \preceq, l)$, with $\mathsf{max}(\mathsf{m})$ we denote the (multi)set $\oplus_{x \in max(X, \preceq)} l(x)$.

On memories we define two operations: one to add a vertex and another one to remove a vertex. These operations are obviously extended to sets of vertices. Given an element $\mathsf{a} \in \mathcal{O} \times \mathcal{I} \times \mathsf{Name}$ and a set of vertices $Y \subseteq X$, with add we denote the operation that takes a memory $\mathsf{m} = (X, \preceq, l)$, the set of vertices Y and the element a and add a new vertex, labeled with a, which is greater than all the vertex in Y. Formally $\mathsf{add}(\mathsf{m}, Y, \mathsf{a})$ is the memory $\mathsf{m}' = (X \cup \{y\}, \preceq', l')$ where $y \notin X$, $l'(y) = \mathsf{a}$ and $l'(x) = l(x)$ if $x \in X$, and $\preceq'$ is obtained closing transitively and reflexively the relation $\preceq \; \cup\{(y', y) \mid y' \in Y\}$ (though not explicitly stated here, we imagine that the set Y is not empty and is a subset of $max(X, \preceq)$). With remove we denote the operation of removing a vertex x from a memory, thus given a memory $\mathsf{m} = (X, \preceq, l)$, and $x \in X$, with $\mathsf{remove}(\mathsf{m}, x)$ we denote the memory $\mathsf{m}' = (X \setminus \{x\}, \preceq', l')$ where $\preceq'$ and l' are the restriction of $\preceq$ and l respectively to $X \setminus \{x\}$ (though not explicitly stated here, we imagine that only maximal elements are removed). We do need some further notation. Consider a multiset z over $\mathcal{O} \times \{1, \ldots, n\} \times \mathsf{Name}$, and an index $i \in \{1, \ldots, n\}$, with $\lfloor z \rfloor_i$ we denote the multiset defined as follows: $\lfloor z \rfloor_i(a) = z(a)$ if $\mathsf{i}_m(a) = i$ and $\lfloor z \rfloor_i(a) = 0$ otherwise.

The notion of membrane system does not change, it changes however the one of configuration (than now has a memory).

Definition 9. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system. Then a configuration with memory is the pair $\mathcal{C} = (C, \mathsf{m})$ where $C = (w_1, \ldots, w_n)$ is the tuple of multisets over $\mathcal{O}$ and $\mathsf{m} = (X, \preceq, l)$ is a memory such that for each $i \in \{1, \ldots, n\}$ it holds that $w_i = \mathsf{obj}_m(\lfloor \mathsf{max}(\mathsf{m}) \rfloor_i)$.*

The initial configuration $\mathcal{C}_0$ is the pair (C_0, m_0), where $C_0 = (w_1^0, \ldots, w_n^0)$ and $\mathsf{m}_0 = (X, \preceq, l)$ is a memory such that $\forall x \in X$, $\mathsf{rule}_m(l(x)) = \bot$ and $\forall x, y \in X$. $x \preceq y$ implies $x = y$.

Given a configuration with memory $\mathcal{C} = (C, \mathsf{m})$, then $\eta(\mathcal{C})$ is C and $\gamma(\mathcal{C})$ is m.

A configuration has now a memory and the requirement is that for each maximal element of the memory corresponds an object in the membrane configuration. The initial memory is such that the maximal elements carry the information on the rule stating that they have not been produced by any rule, and the partial ordering is the discrete one.

Example 3. Consider the membrane system with just one membrane with the set of rules: $\{r_1^1 : a \to (a, \mathsf{here}), r_2^1 : a \to (2a, \mathsf{here}), r_3^1 : b \to (a \oplus b, \mathsf{here}), r_4^1 : a \oplus b \to (a, \mathsf{here})\}$ and the following initial configuration: $(a \oplus b, (\{v_1, v_2\}, id, l))$, where id is the identity relation on $\{v_1, v_2\}$, $l(v_1) = (a, 1, \bot)$ and $l(v_2) = (b, 1, \bot)$.

The definition of enabling of a multi-rule vector is the same as for membrane systems: it should be checked on the object part of a configuration (which is closely related to the memory).

Definition 10. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system with memory, $\mathcal{C} = (C, \mathsf{m})$ a configuration with memory, and $\overrightarrow{R}$ a multi-rule vector. Then $\overrightarrow{R}$ is* enabled *at $\mathcal{C}$ whenever $\eta(\mathcal{C})\,[\overrightarrow{R}\rangle$, We denote the enabling of $\overrightarrow{R}$ at $\mathcal{C}$ with $\mathcal{C}\,\{\![\overrightarrow{R}\rangle$.*

Consider a memory $\mathsf{m} = (X, \preceq, l)$ and a subset of vertex $Y \subseteq max(X, \preceq)$ with $\widetilde{\mathsf{max}}(Y)$ we denote the multiset $\oplus_{y \in Y} l(y)$.

Given a configuration with memory $\mathcal{C} = ((w_1, \dots, w_n), \mathsf{m})$ and a multi-rule vector $\overrightarrow{R}$, for each rule r such that $\widehat{R}_{\mathsf{index}(r)}(r) > 0$, with $\mathsf{LHS}_m(r)$ we denote the pair $(u_{\mathsf{index}(r)}, Y)$ where $u_{\mathsf{index}(r)} \subseteq w_{\mathsf{index}(r)}$ is such that $\mathsf{lhs}(r) = u_{\mathsf{index}(r)}$, with $w_{\mathsf{index}(r)}$ in $\eta(\mathcal{C})$, and Y is a subset of the maximal elements in $\gamma(\mathcal{C}) = (X, \preceq, l)$ such that $\lfloor\ \widetilde{\mathsf{max}}(Y)\rfloor_{\mathsf{index}(r)} = u_{\mathsf{index}(r)}$.

Once a multi-rule vector $\overrightarrow{R}$ is enabled at a configuration with memory we have to state the effects of the application of a rule r. The idea is now the following: for each object of the multiset *produced* by a rule we add to the memory a new vertex labeled with the object, the membrane index it belongs to, and the name of rule r.

Consider a rule r enabled at a configuration $\mathcal{C}$, and consider $\mathsf{LHS}_m(r) = (u_{\mathsf{index}(r)}, \{Y\})$. Consider now $\mathsf{rhs}(r)$, and take $\mathsf{rhs}(r)|_\alpha$ with $\alpha \in \{\mathsf{here}, \mathsf{out}\} \cup \{\mathsf{in}_i \mid \mathsf{father}(i) = \mathsf{index}(r)\}$. Then $\mathsf{RHS}_m(r)_i$ is the multiset in $\mathcal{O}$ defined as usual as $\mathsf{Rhs}(r)_i$, and the new memory is obtained from $\gamma(\mathcal{C}) = (X, \preceq, l)$ by adding for each object o in $\mathsf{RHS}_m(r)_i$ a new vertex y greater than any vertex in Y and labeled with $(o, i, \mathsf{name}(r))$. We denote this operation as $\mathsf{Add}(\gamma(\mathcal{C}), \mathsf{RHS}_m(r)_i, Y)$ and it is the extension of the operation add defined previously. Given a multi-rule vector $\overrightarrow{R}$, for each i between 1 and n, with overloading of notation, we denote with $\mathsf{LHS}_m(\overrightarrow{R})_i$ the multiset of pairs over $\mathcal{O}$ and set of subsets of indices, defined as $\bigoplus_{r \in R_i} \widehat{R}_i(r) \cdot \mathsf{LHS}_m(r)$ (where the sum for pairs acts as the sum on the multiset part and union on the other), and the tuple of these pairs is denoted with $\mathsf{LHS}_m(\overrightarrow{R})$, $\widehat{\mathsf{LHS}_m}(\overrightarrow{R})$ is the tuple obtained considering only the first components of $\mathsf{LHS}_m(\overrightarrow{R})$ (thus $\mathsf{Lhs}(\overrightarrow{R})$), and $\overbrace{\mathsf{LHS}_m}(\overrightarrow{R})$ is the set of subsets of vertices and it is such that $\forall Y, Y' \in \overbrace{\mathsf{LHS}_m}(\overrightarrow{R})$, $Y \neq Y'$ implies that $Y\mathcal{Y}' = \emptyset$ (all the involved vertices are distinct). Similarly, for each membrane, we denote the effects by $\mathsf{RHS}_m(\overrightarrow{R})_i$ and $\mathsf{RHS}_m(\overrightarrow{R})$ denotes the tuple of these effects and on memory is $\mathsf{Add}(\gamma(\mathcal{C}), \mathsf{RHS}_m(\overrightarrow{R}), \overbrace{\mathsf{LHS}_m}(\overrightarrow{R}))$ where $\overbrace{\mathsf{LHS}_m}(\overrightarrow{R})$ is a *set* of subset of the maximal elements in $\gamma(\mathcal{C})$ that have to be followed by the new objects (thus there is a set of maximal elements for each applied rule).

Definition 11. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \dots, w_n^0,\ R_1,\ \dots,\ R_n)$ be a membrane system with memory, $\mathcal{C}$ a configuration with memory, and $\overrightarrow{R}$ a multi-rule vector such that $\mathcal{C}\,\{[\overrightarrow{R}\rangle$, and assume that $\overbrace{\mathsf{LHS}_m}(\overrightarrow{R})$ is the list of maximal elements of $\gamma(\mathcal{C})$ as described above. Then $\mathcal{C}\,\{[\overrightarrow{R}\rangle\mathcal{C}'$ where $\mathcal{C}'$ is obtained by $\mathcal{C}$ as follows: for each membrane index i, $w_i' = w_i \ominus \mathsf{Lhs}(\overrightarrow{R})_i \oplus \mathsf{Rhs}(\overrightarrow{R})_i$ and the memory is $\mathsf{Add}(\gamma(\mathcal{C}), \mathsf{RHS}_m(\overrightarrow{R}), \overbrace{\mathsf{LHS}_m}(\overrightarrow{R}))$.*

The definition is rather obvious: for each object o produced in a membrane i by the rule $\mathfrak{r}$ a new vertex is added in the memory which is greater than the elements *consumed* by the rule.

Observe that the elements added to the configuration are precisely among the maximal elements in the memory.

Proposition 2. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system with memory, $\mathcal{C}$ a configuration with memory, and $\overrightarrow{R}$ a multi-rule vector such that $\mathcal{C}\,\{[\overrightarrow{R}\rangle \mathcal{C}'$. Take $Y = \mathsf{max}(\gamma(\mathcal{C}'))$ and consider $l(Y)$ which can be seen as a multiset over $\mathcal{O} \times \{1, \ldots, n\} \times \mathsf{Name}$. Then for each $i \in \{1, \ldots, n\}$. $\mathsf{obj}_m(\lfloor l(Y) \rfloor_i) = w'_i$ where $\eta(\mathcal{C}') = (w'_1, \ldots, w'_n)$.*

Example 4. Consider the membrane system of Example 3. At the initial configuration the following sets of rules are enabled: $\{r_1^1 \oplus r_3^1\}$, $\{r_2^1 \oplus r_3^1\}$, $\{r_4^1\}$. Consider the last one. The execution of it gives the configuration $((a, r_4^1), (\{v_1, v_2, v_3\}, id \cup \{(v_1, v_3), (v_2, v_3)\}, l'))$ where $l'(v_1) = l(v_1), l'(v_2) = l(v_2)$ and $l'(v_3) = (b, 1, r_4^1)$.

Performing another one, for instance $\{r_1^1 \oplus r_3^1\}$, would give a different memory.

We show that this is a conservative extension of membrane systems, as to each step in a membrane system with memory, a step corresponds in the membrane system where all the added information is forgotten.

Proposition 3. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system, $\mathcal{C}$ a configuration with memory, $\overrightarrow{R}$ a multi-rule vector such that $\mathcal{C}\,\{\overrightarrow{R}\rangle$, and let $\mathcal{C}\,\{[\overrightarrow{R}\rangle \mathcal{C}'$. Then $\eta(\mathcal{C})\,[\overrightarrow{R}\rangle$ and $\eta(\mathcal{C})\,[\overrightarrow{R}\rangle\,\eta(\mathcal{C}')$.*

We discuss now when a rule r can be *reversely* applied in this setting. Again the intuition is rather simple, just check if there are enough objects bearing the name of the rule r among the maximal elements of the memory. Let m be a memory, $\mathfrak{n}$ be a rule name, and i a membrane index, then with $\blacktriangleleft^i_{\mathsf{name}(r)}(\mathsf{m})$ we denote the multiset on $\mathcal{O}$ defined as

$$\blacktriangleleft^i_{\mathsf{name}(r)}(\mathsf{m}) = \bigoplus_{x \in \mathsf{max}(\mathsf{m})} \{\mathsf{obj}_m(l(x)) \mid \mathsf{rule}_m(l(x)) = \mathsf{name}(r) \ \wedge \ \mathsf{i}_m(l(x)) = i\}$$

Let r be a rule and $\mathcal{C}$ be a configuration of a membrane system with memory Π_m. Then r is reversely enabled at $\mathcal{C} = ((w_1, \ldots, w_n), \mathsf{m})$ whenever, for all $k \in \mathcal{I}(r)$, $\mathsf{rhs}(r)_k \subseteq \blacktriangleleft^k_{\mathsf{name}(r)}(\mathsf{m})$. The reverse enabling is summarized in the following definition.

Definition 12. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system with memory, $\mathcal{C}$ a configuration, and $\overrightarrow{R}$ a multi-rule vector. Then $\overrightarrow{R}$ is* reversely enabled *at $\mathcal{C}$ if for rule r in $\overrightarrow{R}$ $\widehat{R}_{\mathsf{index}(r)} \cdot \mathsf{rhs}(r)_k \subseteq \blacktriangleleft^{w_k}_{\mathsf{name}(r)}(\gamma(\mathcal{C}))$. The reverse enabling of a multi-rule vector is denoted with $C\,\langle\overrightarrow{R}]\}$.*

In this case we have to find, for each each instance of a given rule, enough objects produced by an instance of the same rule at the same (local) configuration.

Once a multi-rule vector is reversely enabled, it may be applied. We start showing what it means to undo a single rule r. Given a configuration $\mathcal{C}$, with the memory $\gamma(\mathcal{C}) = (X, \preceq, l)$, for each index $k \in \mathcal{I}(r)$ we have that $\mathsf{rhs}(r)_k$ is contained in $\blacktriangleleft^k_{\mathsf{name}(r)}(\gamma(\mathcal{C}))$. Consider a subset $Y \subseteq max(X, \preceq)$ such that $\mathsf{obj}_m(\lfloor \oplus_{y \in Y} l(y) \rfloor_k) = \mathsf{rhs}(()r)_k$, then what we have to do on the memory is just to remove the set Y from the memory. The set of these vertices are denoted with $\widehat{\mathsf{RHS}_m}(r)$ and it extends obviously to $\overrightarrow{R}$. Clearly we require that these sets of vertices are disjoint.

Definition 13. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system with memory, $\mathcal{C} = ((w_1, \ldots, w_n), \mathsf{m})$ be a configuration, and $\overrightarrow{R}$ a multi-rule vector such that $C \,\langle\overrightarrow{R}]\!\}$. Then $\mathcal{C}' = ((w_1', \ldots, w_n'), \mathsf{m}')$, where $w_i' = w_i \ominus \mathsf{Rhs}(\overrightarrow{R})_i \oplus \mathsf{Lhs}(\overrightarrow{R})_i$ and m' is obtained from m by removing all the vertex in m corresponding to the object in $\mathsf{RHS}_m(\overrightarrow{R})$, thus $\mathsf{m}' = \mathsf{remove}(\mathsf{m}, \overbrace{\mathsf{RHS}_m}(\overrightarrow{R}))$, is the configuration reached by reversely executing $\overrightarrow{R}$ at $\mathcal{C}$. As before it is denoted with $C \,\langle\overrightarrow{R}]\!\}\, C'$.*

Example 5. Consider the membrane system of Example 3 and the computation step done in Example 4. The set $\{r_4^1\}$ is reversely enabled and

$$(b, (\{v_1, v_2, v_3\}, id \cup \{(v_1, v_3), (v_2, v_3)\}, l')) \;\; \langle\{r_4^1\}]\!\} \;\; (a \oplus b, (\{v_1, v_2\}, id, l))$$

where the labeling are those in Example 3 and 4.

Consider another membrane system with just one membrane with the set of rules: $\{r_1^1 : b \to (a \oplus b, \mathsf{here})\}$ and the initial configuration (b, m_0). Applying to this configuration $\{r_1^1\}$ we have

$$(b, \mathsf{m}_0) \,\{\![\{r_1^1\}\rangle\, (a \oplus b, \mathsf{m}_1)$$

where $\mathsf{m}_1 = (\{v_1, v_2, v_3\}, \preceq, l)$ where $v_1 \preceq v_2, v_1 \preceq v_3$ and l is the following: $l(v_1) = (b, 1, \bot), l(v_2) = (a, 1, r_1^1)$ and $l(v_3) = (b, 1, r_1^1)$. To this configuration we can apply again the same rule:

$$(a \oplus b, \mathsf{m}_1) \;\; \{\![\{r_1^1\}\rangle \;\; (a \oplus a \oplus b, \mathsf{m}_2)$$

where now m_2 is $(\{v_1, v_2, v_3, v_4, v_5\}, \preceq', l')$ with $v_3 \preceq' v_4, v_3 \preceq' v_5$ and the new vertices are labelled as $l(v_4) = (a, 1, r_1^1)$ and $l(v_5) = (b, 1, r_1^1)$. Reversely applying the unique rule we could have now a choice: either consider the vertices $\{v_4, v_5\}$ or $\{v_2, v_5\}$. In the latter case we have

$$((a \oplus a \oplus b, \mathsf{m}_2) \;\; \langle\{r_1^1\}\}\, (a \oplus b, \mathsf{m}_3)$$

where m_3 is obtained from m_2 by removing the vertices v_2 and v_5. This choice (which is investigated in a different setting in [19]) has as consequence that we cannot further undo going back to the initial configuration.

Clearly if the vertices $\{v_4, v_5\}$ are considered, then the configuration $(a \oplus b, \mathsf{m}_1)$ is obtained again.

Again the loop lemma can be proved also in this setting but, as the previous example points out, it is a weaker version with respect to the one we introduced previously.

Lemma 2 (Loop lemma for membrane system with memory). *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system with memory, $\mathcal{C}$ a configuration, and $\overrightarrow{R}$ be a multi-rule vector such that $\mathcal{C} \,\{\![\overrightarrow{R}\rangle$, and let $\mathcal{C}'$ be the configuration reached by executing $\overrightarrow{R}$,* i.e. *$\mathcal{C} \,\{\![\overrightarrow{R}\rangle\, \mathcal{C}'$. Then there exists a set of vertices in $\gamma(\mathcal{C}')$ associated to the object to be removed by the reverse application of $\overrightarrow{R}$, such that $\mathcal{C}' \,\langle\overrightarrow{R}]\!\}\mathcal{C}''$ and $\mathcal{C} = \mathcal{C}''$.*

Observe that not necessarily the objects consumed by the application of a multi-rule vector are those used in the reverse application of it. Hence we not necessarily obtain again the same memory. However, if the memory is the same, then the vector multi-rule reversely applied is the same we started with.

Theorem 2. *Let Π_m be a membrane system with memory, $\mathcal{C}$ a configuration, and $\overrightarrow{R}$ be a multi-rule vector such that $\mathcal{C}\,\{\![\overrightarrow{R}\rangle\,\mathcal{C}'$. Then for all multi-rule vector $\overrightarrow{R'}$ such that $\mathcal{C}'\,\langle\overrightarrow{R'}]\!\}\,\mathcal{C}$ it holds that $\overrightarrow{R'} = \overrightarrow{R}$.*

Obvioulsy, the reversing in a membrane system with memory and the reversing in the membrane system where the additional information are forgotten, are related in a precise way.

Proposition 4. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system with memory, $\mathcal{C}$ a configuration, and $\overrightarrow{R}$ be a multi-rule vector such that $\mathcal{C}\,\{\![\overrightarrow{R}\rangle$, and let $\mathcal{C}'\,\langle\overrightarrow{R}]\!\}\,\mathcal{C}$. Then $\eta(\mathcal{C}')\,\langle\overrightarrow{R}]\,\eta(\mathcal{C})$.*

5 Future works

Reversibility in membrane systems has several facets. One is connected with determinism and the fact that each configuration has just a single predecessor, another is related to the amount of information needed to *reconstruct* past configurations. Concerning this view of reversibility, we have proposed a way to add all the relevant informations to *undo* steps properly. It must be said that many other solutions are conceivable, depending on the amount of information needed, for instance objects may be enriched to carry the history. The approach we presented here has the characteristic that the memory not only allow to reverse steps properly but also keep tracks of the dependencies among steps and objects.

Beside continuing to investigate on how reversibility can be achieved in membrane systems, we put two possible research issues. Here we have considered that all the rules are reversible, but this assumption is a maybe too strong when computations that are inspired by nature are considered. We may imagine that some rules produce *irreversible* effects, that cannot be undone. This may be modelled simply forgetting the rules names in both approaches. However this opens many questions on how to actually reverse computations and also on the notions of causality as investigated in [20] or [2]. Various situations may be devised in this setting, similarly to what is done in [19]. Here some events are undone but still some of their effects may remain. This idea can be possibly implemented also in membrane systems, opening new interesting feature.

Another issue is the possibility of combining the two ways: a part of the multi-rule vector is used to compute forward, another part is used to undo some effects. Again this has to be fully investigated.

Acknowledgement

The author acknowledge the useful remarks and suggestions by the anonymous reviewers.

References

1. Agrigoroaiei, O., Ciobanu, G.: Reversing computation in membrane systems. Journal of Logic and Algebraic Programming 79(3-5), 278–288 (2010)
2. Agrigoroaiei, O., Ciobanu, G.: Rule-based and object-based event structures for membrane systems. Journal of Logic and Algebraic Programming 79(6), 295–303 (2010)
3. Alhazov, A., Freund, R., Morita, K.: Sequential and maximally parallel multiset rewriting: reversibility and determinism. Natural Computing 11(1), 95–106 (2012)
4. Aman, B., Ciobanu, G.: Computational power of protein interaction networks. In: Mauri, G., Dennunzio, A., Manzoni, L., Porreca, A.E. (eds.) UCNC 2013. Lecture Notes in Computer Science, vol. 7956, pp. 248–249. Springer (2013)
5. Ciobanu, G., Pan, L., Păun, G., Pérez-Jiménez, M.J.: P systems with minimal parallelism. Theoretical Computer Science 378(1), 117–130 (2007)
6. Ciobanu, G., Pinna, G.M.: Catalytic Petri nets are Turing complete. In: Dediu, A.H., Martín-Vide, C. (eds.) LATA 2012. Lecture Notes in Computer Science, vol. 7183, pp. 192–203. Springer (2012)
7. Ciobanu, G., Pinna, G.M.: Catalytic and communicating Petri nets are Turing complete. Information and Computation 239, 55–70 (2014)
8. Danos, V., Krivine, J.: Reversible communicating systems. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. vol. 3170, pp. 292–307. Springer (2004)
9. Freund, R., Kari, L., Oswald, M., Sosík, P.: Computationally universal P systems without priorities: two catalysts are sufficient. Theoretical Computer Science 330(2), 251–266 (2005)
10. Giachino, E., Lanese, I., Mezzina, C.A., Tiezzi, F.: Causal-consistent reversibility in a tuple-based language. In: Daneshtalab, M., Aldinucci, M., Leppänen, V., Lilius, J., Brorsson, M. (eds.) PDP 2015. pp. 467–475. IEEE Computer Society (2015)
11. Ibarra, O.H.: On strong reversibility in P systems and related problems. International Journal of Foundations of Computer Science 22(1), 7–14 (2011)
12. Kleijn, J., Koutny, M.: A Petri net model for membrane systems with dynamic structure. Natural Computing 8(4), 781–796 (2009)
13. Kleijn, J., Koutny, M., Rozenberg, G.: Towards a Petri net semantics for membrane systems. In: WMC 2005. Lecture Notes in Computer Science, vol. 3850, pp. 292–309. Springer (2005)
14. Lanese, I., Mezzina, C.A., Stefani, J.: Reversing higher-order pi. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. Lecture Notes in Computer Science, vol. 6269, pp. 478–493. Springer (2010)
15. Lanese, I., Mezzina, C.A., Stefani, J.: Reversibility in the higher-order π-calculus. Theoretical Computer Science 625, 25–84 (2016)
16. Lanese, I., Mezzina, C.A., Tiezzi, F.: Causal-consistent reversibility. Bulletin of the EATCS 114 (2014)
17. Leporati, A., Zandron, C., Mauri, G.: Reversible P systems to simulate fredkin circuits. Fundamenta Informaticae 74(4), 529–548 (2006)
18. Nishida, T.Y.: Reversible P systems with symport/antiport rules. In: Paun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A. (eds.) WMC 2010. pp. 452–460 (2010)
19. Phillips, I., Ulidowski, I.: Reversibility and asymmetric conflict in event structures. Journal of Logic and Algebraic Methods in Programming 84(6), 781–805 (2015)
20. Pinna, G.M., Saba, A.: Modeling dependencies and simultaneity in membrane system computations. Theoretical Computer Science 431, 13–39 (2012)

21. Păun, A., Păun, G.: The power of communication: P systems with Symport/Antiport. New Generation Computing 20(3), 295–306 (2002)
22. Păun, G.: Computing with membranes: An introduction. Bulletin of the EATCS 67, 139–152 (1999)
23. Păun, G., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
24. Song, T., Shi, X., Xu, J.: Reversible spiking neural P systems. Frontiers of Computer Science 7(3), 350–358 (2013)

Families of Languages Encoded by SN P Systems[⋆]

José M. Sempere

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València,
jsempere@dsic.upv.es

Abstract. In this preliminary work, we propose the study of SN P systems as classical information encoders. By taking the spike train of an SN P system as a (binary) source of information, we can obtain different languages according to a previously defined encoding alphabet. We provide a characterization of the language families generated by the SN P system in this way. This characterization depends on the way we define the encoding scheme: bounded or not bounded and, in the first case, with one-to-one or non injective encodings. Finally, we propose a network topology in order to define a cascading encoder.

Keywords: SN P systems, formal languages, codes, word enumerations.

1 Introduction

Spiking Neural P systems (SN P systems) were proposed as a model that combines some aspects of neural networks and some others from P systems. Basically, they have been proposed as acceptor systems, language generators or (encoded) word transducers. We focus our attention to the generative capacity of this model. Typically, the language generated by the system is taken as the binary words defined by the spike train that the system outputs. This approach was first formulated in [5], and later developed in [1].

In this work, we consider the SN P system as a classical information source that can generate encoded strings as outputs. The binary codes can be established in an exogenous predefined way and, for a fixed encoding alphabet, the system generates a (possibly) infinite language. So, any SN P system can generate different languages depending on the encoding that has been defined. We will overview different situations within this approach: First, for a fixed integer value we will distinguish between one-to-one and non-injective cases. Then, different encoding schemes where the integer value tends to infinity will be overviewed and, finally, a network topology that connect different SN P systems to produce a cascading encoder will be proposed.

[⋆] Part of this work appeared as *Families of Languages Associated with SN P Systems: Preliminary Ideas, Open Problems.* Gh. Păun, J.M. Sempere. Bulletin of the Membrane Computing Society, Issue 2, December 2016, pp 161-164. http://membranecomputing.net/IMCSBulletin/

2 Basic concepts

We consider that the reader knows basic concepts and results from formal language theory, otherwise we refer to [10]. In the same way, we consider that the reader is familiar with the basic concepts and results about P systems and membrane computing, otherwise we refer to [8] and [4].

In what follows, we provide some basic definitions related to Spiking Neural P systems from [4].

Definition 1. *A spiking neural P system (*SN P system*, for short) of degree* $m \geq 1$ *is defined by the tuple* $\Pi = (O, \sigma_1, \sigma_2, \cdots, \sigma_m, syn, in, out)$ *where*

1. $O = \{a\}$ *is the singleton alphabet of* spikes
2. $\sigma_1, \sigma_2, \cdots, \sigma_m$ *are* neurons *of the form* $\sigma_i = (n_i, R_i)$*,* $1 \leq i \leq m$*, where*
   (a) $n_i \geq 0$ *is the initial number of spikes contained in* σ_i
   (b) R_i *is a finite set of rules of the following two forms*
      i. *firing or spiking rules* $E/a^c \to a; d$ *where* E *is a regular expression over* a*, and* $c \geq 1$*,* $d \geq 0$ *are integer numbers. We will omit* E *whenever it be equal to* a^c*, and we will omit* d *if it is equal to 0.*
      ii. *forgetting rules* $a^s \to \lambda$*, for* $s \geq 1$*, with the restriction that for each spiking rule* $E/a^c \to a; d$ *then* $a^s \notin L(E)$ *(*$L(E)$ *is the regular language defined by* E*)*
3. $syn \subseteq \{1, 2, \cdots, m\} \times \{1, 2, \cdots, m\}$ *with* $(i, i) \notin syn$*, for* $1 \leq i \leq m$*, is the directed graph of synapses between neurons;*
4. $in, out \in \{1 \cdots m\}$ *indicate the* input *and the* output *neurons of* Π*.*

At neuron σ_i, the firing rules $E/a^c \to a; d$ are applied as follows: if the neuron contains $k \geq c$ spikes and $a^k \in E$ then c spikes are removed from σ_i and one spike is delivered to all the neurons σ_j connected to σ_i with $(i, j) \in syn$. If $d = 0$ the spike is immediately emitted, otherwise it is emitted after d computation steps (during these computation steps, the neuron is closed, so it cannot receive spikes, it cannot apply the rules and, subsequently it cannot send new spikes). At neuron σ_i, the forgetting rule $a^s \to \lambda$ is applied as follows: if the neuron σ_i contains exactly s spikes and no firing rule can be applied then all the spikes of the neuron are removed.

A configuration of the system at an instant t during a computation is defined by the tuple $(i_1/t_1, \cdots, i_m/t_m)$ that denotes the number of spikes that are at every neuron together with the computation time needed to open the neuron. The initial configuration of the SN P system is $(n_1/0, \cdots, n_m/0)$. During a computation, the moments of time when a spike is emitted by the output neuron will be marked by '1' while the other moments are marked by '0'. The binary sequence that is obtained in such a way during the computation is called the *spike train* of the system. In the sequel, we will omit the input neuron, and we will work with SN P systems as *language generators*.

The language generated by any SN P system depends on the interpretation given to the spike train that it outputs. For any halting computation, we can take the finite spike train as a string over the binary alphabet $B = \{0, 1\}$, or we can take the intervals between output spikes with different approaches such as those described in [9]. In what follows, we will consider the spike train as a generator of binary strings.

For any SN P system Π, the language generated by Π as described before will be denoted by $L_1(\Pi)$.

3 Languages encoded by SN P systems

Our approach to the languages generated by SN P systems is different from the previously referred ones. Actually, the present research idea occurred in a framework related to classical communication channels with encoded information, where, for every SN P system, different languages can be associated to the system depending on a parameter that fixes a time window to analyze the spike train.

For any SN P system Π, we take the binary language $L_1(\Pi)$ and we encode blocks of k digits, for all the positive integer values k, in such a way that languages $L_k(\Pi)$ are obtained. Of course, we have to take care of the case when the spike train is not of a length which is a multiple of the considered k. In this case, we add symbols 0 so that the obtained binary string is of a length divisible by k.

More formally, let $B = \{0, 1\}$ be the binary alphabet, let $k \geq 1$ be a natural number, let B^k be the set of all strings from B whose length is k, and V_k be an alphabet. In general, any alphabet can be considered but we will associate a different symbol for every word in B^k. Consider a mapping $\varphi_k : B^k \longrightarrow V_k$. For each string $w \in B^*$ we consider the string ${}_kw = w0^t$, where $t = \min\{n \geq 0 \mid |w0^n| \text{ is a multiple of } k\}$.

The string ${}_kw$ can be written in the form ${}_kw = x_1x_2 \dots x_s$, such that $|x_j| = k$ for all $j = 1, 2, \dots, s$. Then, φ_k can be extended to $(B^k)^*$ in the natural way: $\varphi_k(y_1y_2 \dots y_t) = \varphi_k(y_1)\varphi_k(y_2) \dots \varphi_k(y_t)$ for all $y_i \in B^k, 1 \leq i \leq t, t \geq 0$. We can see the encoding approach that we have just described in Fig.1.

Thus, for an SN P system Π and an encoding φ_k as above, we can define the language

$$L_{\varphi_k}(\Pi) = \{\varphi_k({}_kw) \mid w \in L_1(\Pi)\}.$$

In what follows, we write $L_k(\Pi)$ instead of $L_{\varphi_k}(\Pi)$. The language $L_k(\Pi)$ depends on the encoding φ_k, hence a family of languages can be associated with Π by varying k and the mapping φ_k. Observe, that the language $L_1(\Pi)$, as defined at the end of section 2, is a particular case of $L_k(\Pi)$ when $k = 1$, given that φ_1 can be trivially defined as the identity mapping. We define the family of languages $F(\Pi) = \{L_k(\Pi) | k \geq 1\}$.

Already at this very general level there appear several research issues. In what follows, we consider two classes of mappings φ_k and investigate the closure properties of the corresponding families of languages generated by SN P systems.

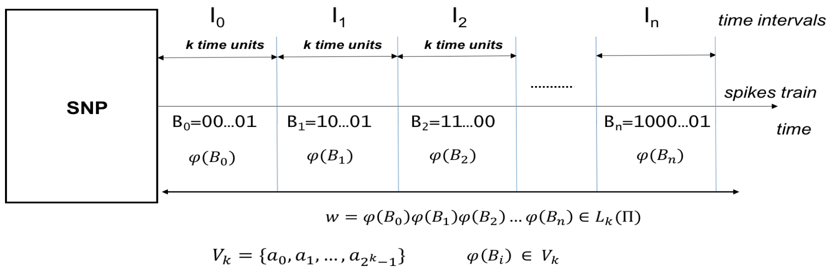


Fig. 1. SN P systems as language encoders: The case of intervals of lenght k.

3.1 The one-to-one case

A natural possibility is to order in a precise way, e.g., lexicographically, the strings in B^k, and to associate with each of them a distinct symbol from an alphabet V_k with 2^k elements, that is, assuming that φ_k is injective.

We can state the following property that arises from the definition of $L_1(\Pi)$ as a finite language.

Property 1. Let Π be a SN P system. Then, if $L_1(\Pi)$ is finite the so are each $L_k(\Pi)$ for $k > 1$.

From the Property 1, we can deduce that if $L_1(\Pi)$ is finite, then the family $F(\Pi)$ is finite, up to a renaming of symbols of alphabets V_k.

Property 2. Let Π be a SN P system. Then, if $L_1(\Pi)$ is infinite, then so are each $L_k(\Pi)$ for $k > 1$.

If $L_1(\Pi)$ is infinite, then $F(\Pi)$ can be an infinite family, because the alphabet of $L_{k+1}(\Pi)$ might be larger than the alphabet of $L_k(\Pi)$. This is the case, for instance, for the SN P system Π generating $L_1(\Pi) = \{1^n01^m \mid n, m \geq 1\}$ (which is an infinite regular language).

The fact that the encoding is one-to-one is rather restrictive: the passing from the binary language $L_1(\Pi)$ to a given $L_k(\Pi)$ can be done by means of a sequential transducer (a gsm, in the usual terminology, [10]). Conversely, the passage from $L_k(\Pi)$ to $L_1(\Pi)$ is done by an one-to-one (non-erasing) morphism, which implies that the converse passage is done by an inverse morphism. This observation can be formally formulated as follows.

Proposition 1. *If $L_1(\Pi) \in FL$, where FL is a family of languages closed under gsm mappings or under inverse morphisms, then $L_k(\Pi) \in FL$, for all $k \geq 1$. If FL is closed under non-erasing morphisms and $L_k(\Pi) \in FL$, then also $L_1(\Pi) \in FL$.*

Families as FL above are REG, LIN, CF in the Chomsky hierarchy, hence if $L_1(\Pi)$ is regular, linear or context-free, then so are all languages $L_k(\Pi)$, and conversely.

This means that each family $F(\Pi)$ contains only languages of the same type in the Chomsky hierarchy (for instance, it is not possible to have a context-free non-regular language $L_k(\Pi)$ together with a regular language $L_j(\Pi)$, for some $k \neq j$.

3.2 The non-injective case

The previous type-preserving Proposition 1 does not hold in the case of using encodings which are not one-to-one.

Here is an example: Consider Π such that $L_1(\Pi) = \{1^n01^n \mid n \geq 1\}$ (SN P systems are universal, [5], hence any language can be taken as the starting language). Of course, $L_1(\Pi)$ is context-free non-regular.

Consider the encoding $\varphi_k : B^k \longrightarrow \{a, b\}$ defined by $\varphi_k(w) = a$ if $|w|_0 \leq 1$, and $\varphi_k(w) = b$ if $|w|_0 \geq 2$. We get

$$L_k(\Pi) = a^+ \cup a^*b, \text{ for } k \geq 4,$$

and

$$L_k(\Pi) = a^+ \cup a^+b, \text{ for } k = 2, 3.$$

Clearly, the languages $L_k(\Pi), k \geq 2$, are regular, in spite of the fact that $L_1(\Pi)$ is (context-free) non-regular.

The properties of the encoding is crucial for the properties of the obtained language families (this is true also in other frameworks, see, e.g., [3] and its references), hence this issue deserves further research efforts.

4 The unbounded case

In the previous section, an encoding of the languages based on blocks of length k has been considered. Now, we consider the limit case, when every string from $L_1(\Pi)$ encodes a different string while k tends to ∞.

Formally, we consider an alphabet $\Sigma = \{a_0, a_1, \cdots, a_p\}$, and the ordered set of strings $\Sigma^* = \{w_0, w_1, \cdots w_i, \cdots\}$. We define the encoding $\varphi_{int} : B^* \longrightarrow \Sigma^*$ such that for every binary string x, $\varphi_{int}(x) = w_{int(x)}$ where $int(x)$ is the integer value of x by taking x as a binary number. The encoding scheme over the SN P system is showed at Fig.2.

For a given alphabet Σ and an application $\varphi_{int} : B^* \longrightarrow \Sigma^*$, we can define the encoded language of any SN P system, as we have described before, as follows

$$L_\infty(\Pi) = \{w \in \Sigma^* \mid \exists x \in L_1(\Pi) \text{ such that } w = z_{int(x)}\}$$

Observe, that Σ^* must be ordered within a precise enumeration of all its words. In this case, the enumeration of the strings in Σ^* (actually, its order) is decisive to preserve the language class from $L_1(\Pi)$ to $L_\infty(\Pi)$.

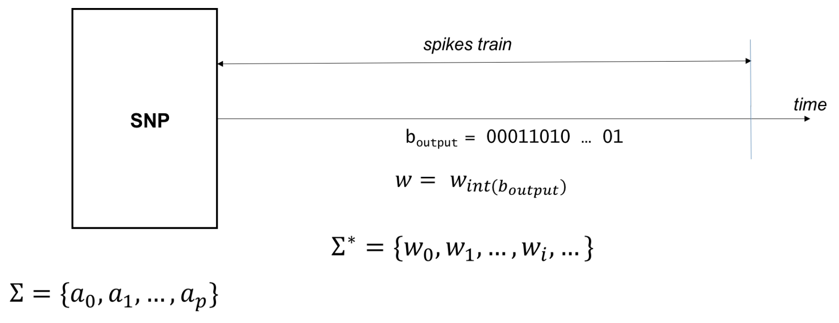

SNP
spikes train
time
b_{output} = 00011010 … 01
$w = w_{int(b_{output})}$
$\Sigma^* = \{w_0, w_1, \ldots, w_i, \ldots\}$
$\Sigma = \{a_0, a_1, \ldots, a_p\}$

Fig. 2. SN P systems as language encoders: The unbounded case.

For example, let us take $L_1(\Pi) = \{(01)^n : n \geq 0\}$ that is a regular language that can be generated by an SN P system given that they have been proved to be universal.

Let $\Sigma = \{a, b\}$ and the languages $L_1 = \{a^n b^n \mid n \geq 0\}$ and $L_2 = \Sigma^* - L_1$. We consider that $L_1 = \{x_1, x_2, x_3 \cdots\}$ and $L_2 = \{y_1, y_2, y_3, \cdots\}$ are lexicographically ordered.

We can define the following enumeration over $\Sigma^* = \{z_1, z_2, \cdots, z_i, \cdots\}$, where

1. If $i \mod 2 = 0$ then $z_i = y_{\frac{i}{2}} \in L_2$ (*even indexes*)
2. If $i \mod 2 = 1$ then $z_i = x_{\lceil \frac{i}{2} \rceil} \in L_1$ (*odd indexes*)

Observe that every string $x \in L_1(\Pi) = \{(01)^n : n \geq 0\}$ encodes an odd integer number given that the binary string ends with '1'. Hence, $L_\infty(\Pi)$ is an infinite subset of L_1 given that, for every string x in $L_1(\Pi)$, the string $z_{int(x)}$ occupies an odd position and, subsequently, it belongs to L_1. Hence. $L_1(\Pi)$ is regular while L_∞ is not.

5 Networks of SN P systems as cascading encoders

Finally, we propose a new way of encoding languages by composing a finite number of SN P systems. In this case we propose a topology based on SN P systems with a tissue-like configuration within a bus connection. Our proposal is showed in Fig.3.

We have a finite set of n SN P systems defined in the usual way. We connect them in the following way: every time that the SN P system i halts, its spike train encodes an integer value k_i that is the parameter to encode the language in the SN P system $i + 1$. Hence, a network of SN P systems can be viewed as cascading encoder for languages. If we connect the SN P systems in a bus topology then, for the iterated case, the last system is connected to the first one.

It opens a new framework which is related to previous works on DNA computing and formal languages [6, 7], where iterated transductions were proved to characterize the entire class of recursively enumerable languages.

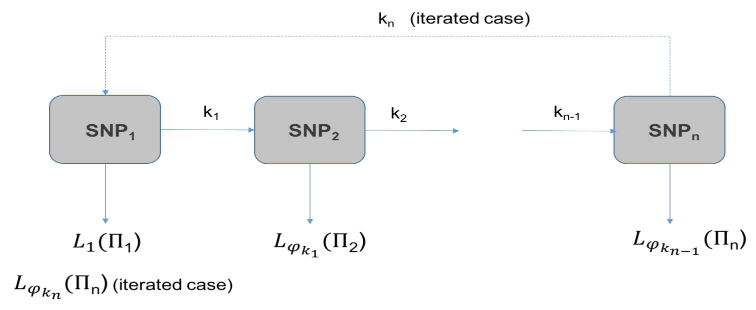

k_n (iterated case)
SNP_1
k_1
SNP_2
k_2
k_{n-1}
SNP_n
$L_1(\Pi_1)$
$L_{\varphi_{k_n}}(\Pi_n)$ (iterated case)
$L_{\varphi_{k_1}}(\Pi_2)$
$L_{\varphi_{k_{n-1}}}(\Pi_n)$

Fig. 3. A network of SN P systems generates a family of languages.

6 Final Comments and future research

The idea of associating a family of languages with a given P system is rather natural. We have illustrated it here with the case of SN P systems, but the same strategy can be applied for any type of P systems producing a language (such that cell-like P systems with external output, SN P systems generating trace languages [2], etc.).

A more systematic study of this idea is of interest, starting with relevant examples, continuing with "standard" formal language theory questions, and ending with possible applications of this approach (as languages generated by the same P system are "genetically" related, maybe in this way one can capture biological connections/dependencies or other types of relationships).

More precisely, we enumerate the following questions related to our proposal:

1. We have described a way to encode languages within SN P systems. Now, the reverse problem arises in order to decode languages from the spike train. Here, from an spike train we should obtain the set of binary spike trains that encode it. This issue should be studied in order to complete a classical communication framework.
2. With respect to the encoding properties, we have overviewed only the aspects related to the (non)injective property. Different properties from code theory should produce new results that connect formal language theory, SN P systems and communications systems.
3. The last issue that we have proposed opens different problems related to it. If a network of SN P systems is proposed then we should study the effects of the network topology and the number of SN P systems over the families of languages. In this sense, the number of SN P system could be considered a descriptional complexity measure.

These aspects and new ones will be reported in future works.

References

1. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez. On string languages generated by spiking neural P systems. *Fundamenta Informaticae*, 75, 1-4 (2007), pp. 141-162.
2. H. Chen, M. Ionescu, A. Păun, Gh. Păun, B. Popa: On trace languages generated by spiking neural P systems, *Eighth International Workshop on Descriptional Complexity of Formal Systems* (DCFS 2006), June 21-23, 2006, Las Cruces, New Mexico, USA, pp. 94–105.
3. E. Csuhaj-Varjú, G. Vaszil: On counter machines versus dP automata. *Membrane Computing. 14th Intern. Conf., CMC 2013, Chişinău, August 2013* (A. Alhazov et al., eds.), LNCS 8340, Springer, Berlin (2014), pp. 138–150.
4. O.H. Ibarra, A. Leporati, A. Păun, S. Woodworth: Spiking neural P Systems, in *The Oxford Handbook of Membrane Computing* (Gh. Păun, G. Rozenberg, A. Salomaa, eds.) Oxford University Press, 2010.
5. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), pp. 279–308.
6. V. Manca: On the generative power of iterated transduction. In *Words, Semigroups, & Transductions* (M. Ito, Gh. Păun, S. Yu, eds.) pp. 315-327. World Scientific, 2001.
7. V. Manca, C. Martín-Vide, Gh. Păun: New computing paradigms suggested by DNA computing: computing by carving. *BioSystems* 52 (1999) pp. 47-54.
8. Gh. Păun. Membrane Computing. An Introduction, Springer, 2002.
9. Gh. Păun, M.J. Pérez Jiménez, G. Rozenberg. Spike trains in spiking neural P systems. *International Journal of Foundations of Computer Science*, 17 4 (2006) pp. 975-1002.
10. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 vols., Springer-Verlag, Berlin, 1997.

On the Robust Power of Morphogenetic Systems for Time Bounded Computation

Petr Sosík[2], Vladimír Smolka[2], Jan Drastík[2], Jaroslav Bradík[2], and Max Garzon[1]

[1] Computer Science, The University of Memphis, Tennessee, USA
[2] Research Institute of the IT4Innovations Centre of Excellence, Faculty of Philosophy and Science, Silesian University, Opava, Czech Republic

Abstract. The time appears ripe to enrich the original idea of membrane computing with principles of self-assembly in space. To this effect, a first step was taken with the introduction of a new such family of models **M systems** (for *morphogenetic system*) that own a number of basic macro-properties exhibited by higher living organisms (such as self-assembly, cell division akin to mitosis and self-healing), while still only leveraging local interactions of simple atomic components and explicit geometric constraints of their constituing elements. Here we further demonstrate that, experimentally *in silico*, **M systems** are in general also capable of demonstrating these properties robustly after being assembled from scratch from some atomic components and entering a homeostatic regime. The results are obtained through a series of experiments carried out with an M system simulator designed to implement this kind of model by researchers interested in exploring new capabilities. We further define probabilistic complexity classes for M systems and we show that the model is theoretically capable of solving **NP**-complete problems in **P**-time, despite apparent problems of an implementation, such as kinetic and concentration bottlenecks.

1 Introduction

The relationship between the macrosciences (such as biology) and the microsciences (such as quantum mechanics and physics) has been a topic of increasing interest for decades. In a pioneering work, Schrödinger [20] explored this connection and pointed to the future developments of a molecular basis for biology, later fully validated by the discovery of the structure of DNA [25] in the 1950s, the development of biotechnology in the 1980s and the genome projects (HGPs, `www.ornl.gov`) of the 1990s. Subsequently, the informatics of biology has been pushed to the forefront by extensive work in $*$-omics in the field of bioinformatics in the 21st century. Bioinformatics can be broadly characterized by the appplication of computer science methods to address biological problems, primarily from the point of view of the science and management of large amounts of data by efficient algorithms. This approach does not address the alternative and more fundamental question of whether biological processes are in essence, at some level, fundamentally information processors, or at least can be understood from that perspective. Turing's paper [24] is perhaps the original most famous attempt at a positive answer, by suggesting a model that would explain why the patterns in a leopard skin exhibit the morphogenesis and resilience to injuries typical in biological organisms.

The primary unit in biological sciences is an *organism* and its fundamental characteristic is *reproduction.* A fundamental distinction between biology and the other natural sciences can be formulated as follows: while physics and chemistry, for example, are governed by interactions that appear immutable and perennial over time, a biological organism is conceived by the physics and chemistry of the world, undergoes a growth process that turns it into an idiosyncratic adult, but eventually dies back into the material world. In the process, the organism produces offspring that inherit some of its uniqueness and perpetuate it over time, but in a very mutable way that creates some sort of living memory and gives rise to evolution. Understandably, the significance of the answers and the complexity of evolution have led computer scientists, and perhaps even biologists, to focus their work on the latter (primarily, natural selection and $*$-omics), which has resulted in relatively poor attention devoted to the organisms themselves, e.g., the morphogenetic growth processes, which may nonetheless play an equally important role in the adult organism itself. A major aim of this work is to focus on models of morphogenesis and the transition into what we term *homeostasis*, i.e., a sustainable, balanced functioning state as a "productive" organism.

There have been two major avenues to address this question, namely membrane computing and virtual cells [23]. The original inspiring idea of *membrane computing*, now usually referred to as P systems [17], was to develop models that could begin to shed light on the role of membranes in the process of morphogenesis of the living cell, while obtaining new insights and approaches to solving difficult problems in computer science. A survey of membrane computing (see [18]) shows a number of works hinting at this kind of model. [11] studies synchronized colonies of membrane-inspired agents, including their behavioural robustness in cases of agent loss or rule failure. A *Spatial P system* embedded in a 2D lattice, partly resembling cellular automata, appeared in [4]. The model was later applied to simulate the collective formation and movement of herring schools [3]. The same authors introduced the *Spatial Calculus of Looping Sequences* (Spatial CLS) [2], assigning to membranes exclusive positions in 2D/3D space. Membrane systems allowing self-assembly of graphs were studied in [7, 6, 5]. A model of morphogenesis of a multicellular body based on abstract membranes displacement and attachment in 3D space was presented in [13] and applied to simulate the growth of colonies of *Dictyostelium discoideu.* Finally, [1] relates membrane systems to 2D *finite interactive systems* representing 2D regular languages. However, all these models assumed an abstract cell as an atomic assembly unit of an abstract nature. Here, we are interested in exploring the developmental process from scratch, i.e., through self-assembly of 1D or 2D primitives allowing for self-assembly of 3D cell-like forms. To be sure, we are not interested in cloning biological organisms (an exercise that sheds little understanding of the key mechanisms at play), but in a deeper examination of potential mechanisms or strategies whereby they may be achieved through a complexification process distributed in space and time, emerging from the bottom-up through local interactions among atomic components naturally available in an environment. Specifically, the objective is to explore higher functions such as internal dynamical homeostasis, self-reproduction, self-healing, for example, and their relationships. (We must point out that, to the best of our knowledge, the actual etiology of these process in biology is not fully known, but even if it were, knowledge of such mechanisms or strategies may prove useful both within biology and other fields such as artificial or extraterrestrial life.)

Perhaps the most appealing feature of membranes is that they bring into the picture an obvious but most fundamental ingredient in the formation of a biological cell, namely the walls that separate it from the external world or the various parts of it. Less known is the more general and primary role of other spatial relationships and constraints in the organization of biological systems, let alone the role of *geometric shape*. An attempt at a general approach to formalization of spatial and geometrical interaction in complex (biological) systems is the 3π calculus [10] based on process algebra.

Recent research points to an increasingly important role in biological morphogenesis of topological and geometric features such as crevices and wrinkles (see e.g., [22].) Another example of current interest is the formation of the mammalian brain cortex. Mechanical and biochemical models have been used. Mechanical models hypothesize that gyris (foldings) in the brain are the results of anisotropic differential growth, while numerical solutions to chemically reaction-diffusion (RD) systems have produced qualitatively approximate patterns in cortex formation, both in 2D and 3D models. Genetic factors, particularly the protein β-catenin recently, are also implicated in the process. These models can be used for prognosis of brain malformations during development in terms of coefficients in the RD model (e.g., polymicrogyria and lissencephaly.) Biologists are also now beginning to discover the importance of the role of even more elementary physical phenomena, such as electric fields and chemical gradients, including their role in chemical signalling in the living cell [19], e.g. in critical mechano-sensitive channels [8].

Simultaneously and from a separate direction, computational ideas from the field of DNA Computing have developed models and theories of DNA self-assembly that capture more directly a “morphogenetic” process of sorts in the form of models of self-assembly of patterns and families of patterns and afford clues as to the nature of and capabilities of morphogenesis [12]. However, once again, these models do not directly afford new knowledge on the fundamental biological problem of morphogenesis and homeostasis that would bring them anywhere near the kind of contribution that other models in natural sciences like physics and chemistry provide us about motion and matter transformation.

Inspired by these developments, the time appears ripe to hybridize P systems and geometric self-assembly in order to explore models of morphogenesis and homeostasis, balancing three somewhat conflicting properties to the best degree possible: biological realism, physical-chemical realism and computational realism. To achieve physical-chemical realism, very critical components and the corresponding dynamic process occurring in a living cell will be specifically represented in the model by appropriate data structures and algorithmic interactions. To achieve computational realism, all components and processes must be modeled at the appropriate level of granularity in both time and resources in order to maintain the computational feasibility of the model. To achieve biological realism, the aggregate observables accumulated over time and space in the model must reflect, to some degree, the corresponding macroscopic observables, e.g., must reflect to some scale or level of granularity known properties of biological organisms at the observable (nano, micro or macro) level, independently of whether they faithfully describe factual processes in biological organisms.

Therefore, the desirable features of the model are self-assembly, self-controlled growth and emerging global behavior that is consistent with observable properties of biological organisms, but which arise from nondeterministic local interactions of elementary components, also consistent with self-assembly and P systems. Towards this goal, we introduced a new such model, **M** systems, in [21], where we also showed its computational universality in the Turing sense. The model is summarized in Sec. 2 to make this paper self-contained. In Sec. 3, we discuss arguments that show how these properties may be guaranteed or to what extent, including a theoretical result and experimental evidence that these properties actually do emerge with very high probability, and provide a characterization of their behavior probabilistically. Sections 4 and 5 complete the view by first defining families of Monte Carlo M systems, and then demonstrating their computational power under a set of restrictions from the perspective of traditional complexity theory. Finally, in Sec. 6, we present some discussion on the significance of the model, some of its implications, and some interesting further problems that could be addressed with plausible extensions of it or experimentation with a simulator.

2 M systems

In this section we briefly introduce morphogenetic (M) systems, basically following the more detailed description in [21]. The reader is referred to [21] or to web sources `sosik.zam.slu.cz/msystem.html` or `bmc.memphis.edu/cytos` for further information.

As mentioned above, introducing geometric features in P systems is a natural and interesting idea of its own. First, it is an intriguing question that may help realize the potential of the original idea of membrane computing, as spatial arrangement is critical for information processing in living cells, colonies, tissues and organisms. Second, it may also further our understanding of computation beyond the scope of traditional computer science, where shape and geometry are not native concepts, but rather that require enormous amounts of effort to build back in, while on the other hand, our understanding of the world is inherently dependent on it. Besides being able to compute in the Turing sense, a model should be able to interact with and "sense" its physical environment, so as to be capable of self-modification and unenthropical evolution, i.e., to increase its fitness (however defined) in its embedding environment.

A primary biological carrier of shape is a protein. This feature is explicitly used in P systems with proteins on membranes [15, 16]. The **M system** extends this concept with explicit geometric features and self-assembly capabilities. The whole system is embedded in an nD Euclidean space $\mathbb{R}^n$ There are three types of objects present in the system: *proteins*, *tiles* and *floating objects.*

Floating objects are small shapeless atomic objects floating freely within the environment, but having at each moment their specified position in space. They can pass through protein channels and participate in mutual reactions with other types of objects, in *discrete time steps.*

Tiles have their pre-defined size and shape, together with specified position and orientation in space at each moment. Tiles can stick together along their edges or at selected points. These edges/points are called *connectors* and they are covered with *glues.* Their connection is controlled by a pre-defined *glue relation.* Thus the tiles can self-assemble into interconnected structures.

Proteins are placed on tiles and, apart from acting as protein channels letting floating objects pass through, they also catalyze their reactions.

Unlike current models of membrane systems, *membranes* are not present even implicitly, but they can only be formed of tiles during the evolution of the M system. Therefore, at the beginning of the evolution, typically *no membranes* are present and they must be subsequently self-assembled. The connected tiles can be also disconnected and/or destroyed under certain conditions. The following definitions provide the elements to capture these properties in a formal model (they can be skipped without hindering understanding of Sec. 3).

2.1 Polytopic tiling

The cornerstone of our concept of morphogenetic self-assembly is an nD tile shaped as a bounded convex polytope (n-polytope) [29], with faces of dimension $n-1$ called *facets.* Hence, a 1D tile is a segment/rod whose facets are its endpoints, a 2D tile is a convex polygon with its edges as facets, a 3D tile is a convex polyhedron with polygons as facets, and so forth. We usually describe a polytope by an ordered list of its vertices.

Furthermore, tile may contain connectors defining its connection to other tiles. Let G be a finite set of *glues.* A *connector* of a tile based on an n-polytope $\mathit{\Delta}$ is a triple $(\mathit{\Delta}_c, g, \varphi)$, where

$\mathit{\Delta}_c \subset \mathit{\Delta}$ is a bounded convex k-polytope where $0 \leq k < n$,
$g \in G$ is a glue,
$\varphi \in (-\pi, \pi)$ is the connecting angle.

We distinguish

– *facet connectors* with $k = n - 1$ where $\mathit{\Delta}_c$ is a facet of the polytope $\mathit{\Delta}$;
– *non-facet connectors* with $k \leq n - 1$ placed anywhere on the surface of the tile.

Two or more connectors can share the same position on a tile. Formally, an *n-dimensional tile* is defined as

$$t = (\mathit{\Delta}, \{c_1, \ldots, c_k\}, g_s), \text{ for } k \geq 0,$$

where $\mathit{\Delta}$ is a bounded convex n-polytope, $c_1, \ldots, c_k$ are connectors and $g_s \in G$ is the *surface glue* covering the entire surface of the tile except where connectors are placed.

If an $(n-1)$-dimensional tile embedded in $\mathbb{R}^n$ we denote its two sides by *in* and *out*. By convention, a 2D tile seen from the side *in* has its vertices ordered *clockwise*. A *non-facet* connector with positive connecting angle is placed on side *in*, one with negative angle is placed on side *out*, and one with zero angle can only be located on some facet of the tile.

Definition 1. *A* polytopic tile system *in* $\mathbb{R}^n$ *is a construct* $T = (Q, G, \gamma, d_g, S)$, *where*

Q *is the set of tiles of dimensions* $\leq n$;
G *is the set of glues;*
$\gamma \subseteq G \times G$ *is the glue relation;*
$d_g \in \mathbb{R}_0^+$ *is the gluing distance (assumed to be small compared to the size of tiles);*
S *is the finite multiset of seed tiles from* Q *randomly distributed in space.*

Note that we generalized definitions in the previous paper [21] so that (a) tiles in $\mathbb{R}^n$ can now have dimension from 1 to n, (b) non-facet connectors can now have dimensions from 1 to $n-1$.

Definition 2. *Consider tiles*

t_1 *with a connector* $c_1 = (\mathit{\Delta}_1, g_1, \varphi_1)$, *where* $\mathit{\Delta}_1 = (u_1, \ldots, u_k)$,
t_2 *with a connector* $c_2 = (\mathit{\Delta}_2, g_2, \varphi_2)$, *where* $\mathit{\Delta}_2 = (v_1, \ldots, v_k)$,

for some $k \geq 1$. *Connector* c_2 can connect *to* c_1 *if the following conditions are met:*

1. *both* c_1 *and* c_2 *are both unconnected;*
2. $(g_1, g_2) \in \gamma$;
3. t_2 *can be positioned so that* $u_1, \ldots, u_k$ *match* $v_k, \ldots, v_1$, *in this order;*
4. *at least one of* c_1, c_2 *is a facet connector.*

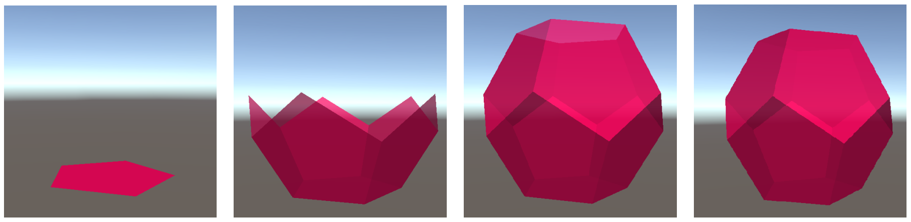
Note that for two 2D tiles connecting with their edges, condition (3) implies the matching of sides *in–in* and *out–out*. As the relation γ is generally non-symmetric, "c_2 can connect to c_1" does not imply that also c_1 can connect to c_2. This is in accordance with natural morphogenetic processes which are often irreversible [9].

Tile t_2 connects to t_1 at angle φ_1, if the connector is $(n-2)$-dimensional. The connecting angle provides a degree of freedom (chosen randomly) to t_2 in the case of k-dimensional connectors, where $k \leq n-3$, on one hand. On the other hand, the connecting angle is not applicable in the case of $(n-1)$D connectors.

If t_2, after its connection, still has a free connector(s) c'_2 now positioned within the distance d_g from a free connector c'_1 on another tile already in place, and either c'_1 can connect to c'_2 or conversely, then they immediately connect together. Similarly, if a free connector of t_2 with a glue g lies within the distance d_g from an existing tile t_3 with surface glue g_s such that $(g, g_s) \in \gamma$, then t_2 connects to the surface of t_3.

Example 1. Consider a polytopic tile system in $\mathbb{R}^3$ with a single glue g and the glue relation $\gamma = \{(g, g)\}$. Let Q contain a 2D tile q shaped as a regular pentagon, with five facet connectors on its edges, each with the glue g and with the connecting angle $\varphi = 2.0345$rad, which is the inner angle between two faces in a dodecahedron. Let finally $S = \{q\}$ be the only seed tile, see the leftmost image. Then, provided that q is available in enough copies, the system assembles as follows.

1. Five tiles q would connect to the five connectors of the seed tile in the first phase, connecting also their five edges starting at vertices of the seed tile as they stick together. The connecting angle determines them to shape as a cup with zig-zag rim with 10 edges (central-left image).
2. another five tiles would connect to these edges, determined by the connecting angle to form an almost-closed shape (central-right image).
3. Finally, the last attached tile encloses the dodecahedral "soccer-ball". All connectors on the tiles match and connect together, hence no further assembly is possible (rightmost image).



2.2 Morphogenetic systems

An M system naturally merges principles of both self assembly and membrane computing. Geometrical structure and growth of each M system is determined by its underlying polytopic tile system. Unlike usual tiling systems, the M system does not assume availability of an unlimited number of copies of each tile. The M system life cycle starts in an initial configuration where only seed tiles are present. Further structures can only be created by the application of rules of the M system.

Formally, for a multiset M we denote by $|M|_a$ the multiplicity of elements a in M. A multiset M with the underlying set O can be represented by a string $x \in O^*$ (by O^* we denote the free monoid generated by O with respect to the concatenation and the identity λ) such that the number of occurrences of $a \in O$ in x represents the value $|M|_a$.

Definition 3. *A morphogenetic system (*M system*) in $\mathbb{R}^n$ (unless stated otherwise, we assume $\mathbb{R}^3$) is a tuple*

$$\mathcal{M} = (F, P, T, \mu, R, r, \sigma),$$

where

$F\ = (O, m, c)$ *is the catalogue of floating objects, where:*
  O *is the set of floating objects;*
  $m : O \longrightarrow \mathbb{R}^+$ *is the mean mobility of each floating object in the environment;*
  $c : O \longrightarrow \mathbb{R}_0^+$ *is the concentration of each floating object in the environment: $c(o)$ copies of object o per spatial unit 1^n;*
P *is the set of proteins;*
$T\ = (Q, G, \gamma, d_g, S)$ *is a polytopic tile system in $\mathbb{R}^n$, with O, P, Q, G all pairwise disjoint;*
μ *is the mapping assigning to each tile $t \in Q$ a multiset of proteins placed on t together with their positions: $\mu(t) \subset P \times \Delta$ where Δ is the underlying polytope of t;*
R *is a finite set of reaction rules;*
$r \in \mathbb{R}_0^+$ *is the reaction radius; a reaction rule can be applied when all objects entering the reaction are positioned within this radius;*
$\sigma : \gamma \longrightarrow O^*$ *is the mapping assigning to each glue pair $(g_1, g_2) \in \gamma$ a multiset of floating objects which are released to the environment within the reaction radius from a new connection with (g_1, g_2), when the connection is established.*

A *reaction rule* from the set R has the form $u \to v$, where u and v are strings containing floating objects, proteins, glues and tiles due to types of rules specified bellow. The necessary condition to apply the rule is that all objects in u are present in the environment within radius r, while certain rules may specify further conditions on the location of objects.

Metabolic rules

Let $u, v \in O^+$ be non-empty multisets of floating objects and $p \in P$ be a protein. The rules containing the symbol [are applicable only when p is placed on an $(n-1)$-dimensional tile, where object to the left of [in the string correspond to the side "out" and those to the right correspond to the side "in" of the tile.

Type	Rule	Effect
simple	$u \to v$	objects in multiset u react to produce v
catalytic	$pu \to pv$	objects in u react in presence of p to produce v;
	$u[p \to v[p$	eventually, u, v must both appear on the side “out”
	$[pu \to [pv$	or on the side “in” of the tile on which p is placed
symport	$u[p \to [pu$	passing of u through protein channel p
	$[pu \to u[p$	to the other side of the tile
antiport	$u[pv \to v[pu$	interchange of u and v through protein channel p

Note that these rules are rather powerful and we will mostly consider some restrictions when studying M systems from the computational power point of view.

Creation rules $u \to t$,

where $t \in Q$ and $u \in O^+$. The rule creates tile t while consuming the floating objects in u. It can be applied if the following holds:

(i) there already exists a tile (say s) in the environment with a free connector c_s such that t can connect to c_s by some of its connectors, and
(ii) floating objects in u exist in the environment within the distance r from c_s.

Then an attempt is made to create tile t and connect it to c_s as specified in Section 2.1. If t would intersect another existing tile, say s', then s' is pushed away to make room for t. This may cause a chain reaction of mutual pushing of tiles in the way. If it is impossible to make enough room for t and t is a polygon, the rule is not applied, otherwise t is shortened so that it just touches s'. Its connector(s) at the shortened end (if any) are preserved.

Destruction rules $ut \to v$,

where $t \in Q$, $u, v \in O^+$. Tile t is destroyed in the presence of the “destructor” multiset of floating objects u. All connectors on other tiles connected to t are released. The objects in u are consumed and the multiset v of “waste” objects is produced.

Division rules $g \overset{u}{—} h \to g, h$,

where $g—h$ is a pair of glues on connectors of two connected tiles, and $u \in O^+$. The rule can be applied when all objects in u are located within reaction distance of the pair of connectors. As a result of application, the two connectors disconnect, while the multiset u is consumed. The connectors remain in their position but they do not reconnect again automatically.

Configuration of the M system is determined by

– positions and Euler angles of all tiles in the environment;
– interconnection graph of connectors on these tiles;
– positions of all floating objects within the finite part of the environment occupied by tiles.

The initial configuration contains only (unconnected) seed tiles in S and a random distribution of floating objects given by their concentration c.

Computation of the M system

The system transits between configurations by application of rules in the set R. At each step, each floating object can be subject to at most one rule, each connector can be subject to at most one creation or division rule, and each tile can be subject to at most one destruction rule.

The rules within each group are chosen nondeterministically until their maximum applicable multiset is obtained, subject to possible trade-offs between rules. Then all the selected rules are applied in parallel to the actual configuration. Finally, each floating object o with mean mobility $m(o)$ changes randomly its position at each step due to the Maxwell-Boltzmann distribution [28] with parameter $a = \sqrt{\pi/8}\, m(o)$ corresponding to Brownian motion of particles in liquid media.

A sequence of transitions of an M system between configurations is called a *computation*. The computation can be finite (if an M system cannot apply any rule, it halts) or infinite, and it is, by definition, nondeterministic. The reader is referred to [21] or to supplementary material available at `sosik.zam.slu.cz/msystem.html` or `bmc.memphis.edu/cytos` for examples.

3 Robust Computational Morphogenesis and Homeostasis

In [21] we have described an demonstrated that M systems are indeed capable of self-assembling from scratch from some atomic components, undergo a process of morphogenesis by the unfolding of the self-assembly rules defined by their local interactions as given by the catalytic, creation and destruction rules, and eventually enter a stable dynamical equilibrium of adulthood in which they will continue to function as long as certain conditions in their environment remain. The system $\mathcal{M}_0$ builds a geometrical structure on two sets of 2D pentagonal tiles: larger tiles self-assembling in a cell-like membrane, and smaller tiles assembling a nuclear membrane. These tiles are much alike that in Example 1 but with different glues on their edges. Some of the larger tiles contain also point connectors on their inner surface, connecting to rod-shaped 1D tiles. Endpoints of rods bear one (straight-oriented) or two (fork-oriented) connectors, allowing the rods to assemble a tree-like structure of cytoskeleton. Specifically, we established that

Proposition 1. *Assuming discrete time and bounded finite resources in the environment, an arbitrary run of the M system $\mathcal{M}_0$ crosses a critical time at which it stops growing and enters a period of homeostasis, where it will remain in functional equilibrium despite certain fluctuations in the environment and/or damage to its internal structure.*

(*Sketch; full description of the M system and more proof details are provided as a supplementary material at*
`sosik.zam.slu.cz/msystem.html` or at `bmc.memphis.edu/cytos`. We summarize the essential part below to make this paper self-contained.)

As pointed out above, discrete time interactions guarantee that at any given time, only a finite number of membranes and objects are contained therein throughout the life of the model, (although they could potentially contain an uncountable number of objects as a continuum.) In the terminology of self-assembly systems, $\mathcal{M}_0$ is locally deterministic and attachment of tiles proceeds as in the aTAM model [26, 27]. As illustrated by Example 1, the geometric structure of the tiles forces them to curve as they are attached and to close upon themselves to eventually form a dodecahedron and present plain geometric blocking for further growth, which thus finishes the membrane building phase when the last keystone tile is attached. Simultaneously an analogous process creates a much smaller nuclear membrane. The attached tiles bear proteins triggering the formation of cytoskeleton by rods, which can grow nondeterministically in various directions from both "poles" of the membrane. Eventually, addition of rods is no longer possible for excluded volume reasons, so the cytoskeleton, and hence morphogenesis is now complete and $\mathcal{M}_0$ enters the "adult" homeostatic phase. Even before this phase is fully completed, the contact of growing rods with the nuclear membrane triggers the process of mitosis which proceeds to create two copies of the cells and separate it into two identical parts, which will then begin a new the entire process and continue while enough supplies and room for growth remain. All this is fully controlled only by local interactions of tiles and floating objects. These properties illustrate how geometry can perform a great deal of work to control the shape of products in self-assembly that could only be performed through other means with great effort, e.g. by hard coding it into the seed, as in the binary counters in the aTAM model [27].

At any point in the morphogenetic process, any "damage" to a configuration of the system $\mathcal{M}_0$ such as knocking off tile that has just been attached, or punching a whole in the membrane) will either simply revert to a previous configuration, or detach a piece of the systems altogether, which will reset it back to a previous state, from which it may further develop as it did before, *perhaps through a different run as it is a nondeterministic system.* Because the stable equilibrium is achieved again with similar characteristics, perhaps the same original individual will not be formed again, but the new individual will bear the characteristic features of the original one. Therefore, the original organism is capable of sustaining certain injuries to some degree of severity to its internal structure, without changing the overall characteristics of the adult organism.

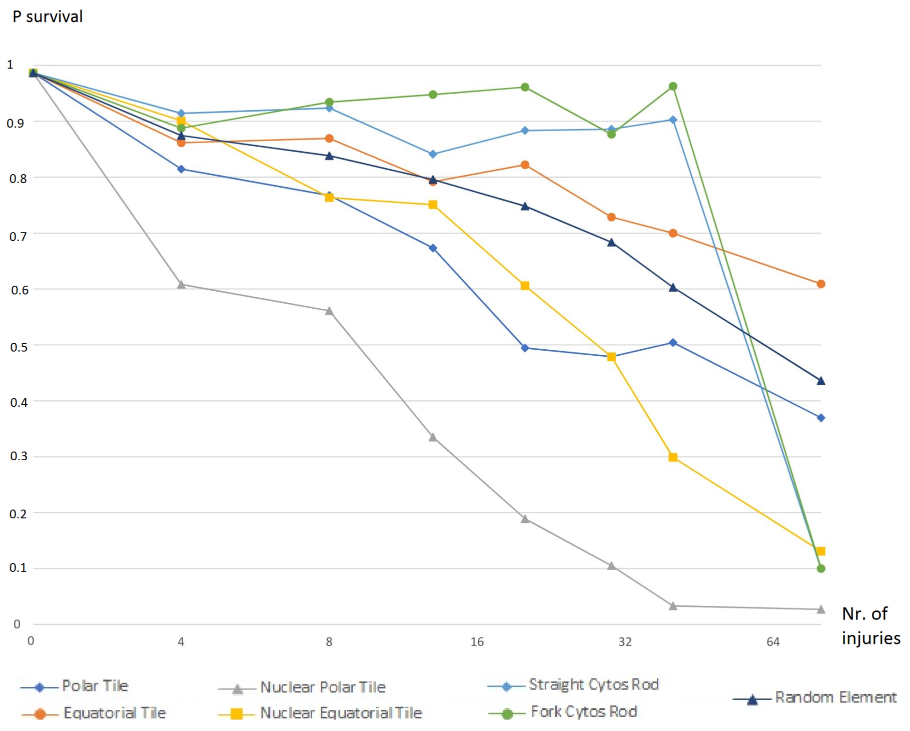


Fig. 1. Self-healing properties of the morphogenetic system $\mathcal{M}_0$ to a number of simultaneous injuries (x-axis) as given by the probability of survival, i.e., to remain in the same and only homeostatic cycle that would have been obtained before the injuries. The estimates have been obtained by 100 runs of the system with injuries inflicted randomly in seven (7) possible ways with up to 80 injuries, as described. The most damaging harm are injuries to nucleus which drop the probability of survival (y-axis) most rapidly. The system is self-healing even when inflicted removal of as many as 40 tiles of any kind, except at nuclear tiles, which reflects the few vulnerabilities of the system. Interestingly, the probability of survival increase with certain number of injuries to cytoskeleton elements with up to 40 rods!

To verify this property quantitatively, we have built an M system simulator (see the link in the previous proof) for arbitrary M systems and run it on $\mathcal{M}_0$ 100 times for 40 iterations causing various injuries to it as described in Fig. 1.

As pointed out in the introduction, computational studies of characteristic biological properties has been an intriguing but poorly addressed subject. Most work has been in the material sciences for simple polymers, gels and even metals, despite the fact that early experimentation in self-asssembly made such properties as self-healing evident [27], with the notable exception of a procedure to re-factor tiles [27] in aTAM systems to build binary counters and the Sierpinski triangle that enable self-healing [27] of holes in partial assemblies in linear time. Here, we generalize the results in Fig. 1 to provide a general definition of self-healing in M systems and demonstrate that a subamily of M systems simultaneously exhibits an additional robustness property characteristic of biological organisms, i.e. a degree of *self-healing.* We first define more precisely the concept of "damage", as follows. From the seed as a root, an M system defines a computation directed graph (digraph) with nodes *all* the possible system configurations obtained from *any* valid computations/runs of the system and with directed arcs all the possible one-step transitions among them. Because the system eventually enters a homeostatic phase, this digraph is finite and can be decomposed into two subdigraphs M_M (for *morphogenetic component*) and M_H (for *homeostatic component.*) The subdiraph consisting of a breadth-first traversal from the root *up to nodes of in-degree 1*, but excluding them (homeostatic) and their transitions leading to their transitive closures, defines the directed subdigraph M_M; M_H is its complementary subdigraph consisting of the remaining nodes (morphogenetic) and arcs. Note that the computation tree is root connected, i.e., every configuration node is accessible from the root by some computation. Thus, every morphogenetic configuration x belongs to a unique maximal homeostatic component (or simply h-component.)

Definition 4. *An* injury *to a morphogenetic system M is a transition of the system given by a pair of configurations (x, y) such that y cannot be obtained from x by a valid application of any one rule of the system M. The degree of the injury is the graph-theoretic distance between x and y. An injury is* sustainable *if both x and y belong to the same h-component. The system is* self-healing *(of degree m, respectively) if and only it can sustain any injury to any homeostatic node x (of degree m, respectively) with probability at least* 50%.

An injury could be caused by an agent in the environment external to M or a malfunction of (the implementation of) the rules of M and may not necessarily destroy any objects in the system, e.g. if y is successor of x in a computation of M. Note that although injuries could cause a transition to a state z that is a not a node in configuration space, the definition implies that the new configuration will not lead to the same homeostatic regime as if no injury had occurred, so that type of injury is never sustainable. Therefore we will only consider injuries as described in Def. 4.

Proposition 2. *Assuming discrete time and bounded finite resources in the environment, every locally deterministic M system is self-healing. In particular, $\mathcal{M}_0$ is self-healing.*

Proof. It is easy to verify that every deterministic system is self-healing, as an injury only amounts to time travel to the past or future. More generally, a locally deterministic system has a unique h-component. □

Conjecture 1. Assuming discrete time and bounded finite resources in the environment, there exist self-healing systems that are also computationally universal.

4 Families of Monte Carlo M systems

Let us denote a *decision problem* X as a pair (I_X, θ_X) where I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total boolean function over I_X. In order to study the computational efficiency of membrane systems, a concept of *recognizer P systems* [14] was introduced. These P systems always halt, and they use specific objects *yes* and *no* such that *exactly one* kind of these objects is produced at the end of each computation. However, halting contradicts the nature of morphogenetic systems whose computation is probabilistic and often reaches a homeostasis where the system stays alive forever (or until external conditions change).

Therefore, we define a *Monte Carlo M system* as follows: it has a distinguished floating object $\texttt{yes} \in O$ such that $c(\texttt{yes}) = 0$ (its concentration in the environment is zero). Its computation is called *accepting* if the object yes is eventually released to the environment, otherwise it is *rejecting.* Furthermore, either at least 1/2 of computations are accepting, or all computations are rejecting. We say that a Monte Carlo M system *accepts in time t* if the object yes is released in first t steps of its arbitrary computation with probability at least 1/2.

Definition 5. *We say that a decision problem $X = (I_X, \theta_X)$ is solvable in a semi-uniform way and randomized polynomial time by a family $\boldsymbol{M} = \{\mathcal{M}(w) \mid w \in I_X\}$ of Monte Carlo M systems if the following holds:*

1. *The family $\boldsymbol{M}$ is* polynomially uniform *by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\mathcal{M}(w)$ from w.*
2. *There exists a polynomial function p, such that for each $u \in I_X$:*
   (a) if $\theta_X(u) = 0$, then every computation of $\mathcal{M}(u)$ is rejecting;
   (b) if $\theta_X(u) = 1$, then $\mathcal{M}(u)$ accepts in time $p(|u|)$.

Let $\mathcal{T}$ be a specific class of morphogenetic systems. We denote by $\mathbf{MRP}^*_{\mathcal{T}}$ the set of all decision problems which can be solved in a semi-uniform way and randomized polynomial time by means of families of M systems from $\mathcal{T}$. Symbol $\mathcal{T}$ can be omitted when general M systems are considered.

5 Computational efficiency

In this section we present a semi-uniform family of Monte Carlo M systems which can probabilistically solve, in a polynomial time, the standard NP-complete problem 3-SAT. The construction and computation of the family is based on the classical strategy of trading space for time, often used in the framework of membrane computing.

Consider a formula $\Phi = C_1 \wedge \ldots \wedge C_m$ in CNF, using the set of variables $\{x_1, \ldots, x_n\}$, for $m, n \geq 1$. We define a polytopic tile system $T_n = (Q, G, \gamma, d_g, S)$ in $\mathbb{R}^2$, where

$Q = \{u_1, \overline{u}_1, v_i \mid 1 \leq i \leq n\} \cup \{w, s_1, s_2, t_b, t_{t1}, t_{t2}, r\}$, where

$u_i, \overline{u}_i$ $1 \leq i \leq n$, are rods of length 2;
v_i $1 \leq i \leq n$, are rectangular 2D tiles of size 2×1;
w is a rod of length 1;
s_1, s_2 are rods of length $2n$;
t_b is a rod of length 2;
t_{t1}, t_{t2} are rods of length 1;
r is a rod of length 5;
$G = \{g_{ib}, \overline{g}_{ib}, g_{it}, g_i, \overline{g}_i, g_{iv}, \overline{g}_{iv} \mid 1 \leq i \leq n\} \cup$
$\{g_p, g_w, g_{1sb}, g_{1st}, g_{2sb}, g_{2st}, g_{1tb}, g_{2tb}, g_{1t1}, g_{1t2}, g_{2t1}, g_{2t1}\}$;
$\gamma = \{(g_p, g_p),\ (g_{nt}, g_{1t1}),\ (g_{1t2}, g_{1st}),\ (g_{1sb}, g_{2tb}),\ (g_{1tb}, g_{2sb}),\ (g_{2tb}, g_{2t1})\} \cup$
$\{(g_{it}, g_{(i+1)b}),\ (\overline{g}_{it}, g_{(i+1)b}) \mid 1 \leq i \leq n-1\} \cup$
$\{(g_{it}, g_w),\ (g_i, g_{iv}),\ (\overline{g}_i, \overline{g}_{iv}),\ (g_{iv}, g_i),\ (\overline{g}_{iv}, \overline{g}_i) \mid 1 \leq i \leq n\}$;
$d_g = 0.1$;
$S = \{u_1\}$.

Let us adopt the convention that a rod of length x has its vertices $(0, x)$, and a rectangular tile of size $x \times y$ has vertices $((0,0), (0,y), (x,y), (x,0))$. Tiles in the set Q have the following connectors:

u_i, $1 \leq i \leq n$, have
- two facet connectors $c_{ib} = (0, g_{ib}, 0)$ and $c_{it} = (2, g_{it}, \varphi_{it})$ (b stands for "bottom" and t for "top"), where $\varphi_{it} = 0$ for $i < n$ and $\varphi_{nt} = \pi/2$;
- two non-facet connectors $c_{i,in} = (1, g_p, \pi/2)$, $c_{i,out} = (1, g_p, -\pi/2)$;
- non-facet 1D connector $c_{i1} = (\langle \frac{3}{2}, \frac{1}{2} \rangle, g_i, \pi/2)$;

$\overline{u}_i$, $1 \leq i \leq n$, have
- two facet connectors $\overline{c}_{ib} = (0, \overline{g}_{ib}, 0)$ and $c_{it} = (2, g_{it}, \varphi_{it})$;
- two non-facet connectors $c_{in} = (1, g_p, \pi/2)$, $c_{out} = (1, g_p, -\pi/2)$;
- non-facet 1D connector $\overline{c}_{i1} = (\langle \frac{1}{2}, \frac{3}{2} \rangle, \overline{g}_i, \pi/2)$;

v_i, $1 \leq i \leq n$, have two facet connectors on their short edges: $c_i = (\langle (0,0), (0,1) \rangle, g_{iv}, 0)$ and $\overline{c}_i = (\langle (1,1), (1,0) \rangle, \overline{g}_{iv}, 0)$;
w has single facet connector $c_w = (0, g_w, 0)$;
s_i, $1 \leq i \leq 2$, have two facet connectors $c_{ist} = (2n, g_{ist}, \pi/2)$ (top) and $c_{isb} = (0, g_{isb}, \pi/2)$ (bottom);
t_b has two facet connectors $c_{tb1} = (0, g_{1tb}, \pi/2)$, $c_{tb2} = (2, g_{2tb}, \pi/2)$;
t_{ti}, $1 \leq i \leq 2$, have two facet connectors $c_{ti1} = (0, g_{it1}, \pi/2)$, $c_{ti2} = (1, g_{it2}, \pi/2)$;
r has a single facet connector $(0, g_p, 0)$.

The set of tiles $\{u_1, \overline{u}_1 \mid 1 \leq i \leq n\}$ represents possible interpretations of variables $x_1, \ldots, x_n$: u_i stands for x_i =true, $\overline{u}_i$ for x_i =false. To simplify the description, let us denote by U_i a tile which is either u_i or $\overline{u}_i$, $1 \leq i \leq n$.

Consider furthermore an M system $\mathcal{M}_\Phi = (F, P, T, \mu, R, r, \sigma)$ in $\mathbb{R}^2$, where:

$F = (O, m, c)$, where:
$O = \{a_1, \ldots, a_6, b_1, \ldots, b_{m+1}, yes\}$, are floating objects;
$m(o) = 2$ for each $o \in O$;
$c(a_4) > 0$ (see proof of Lemma 4 in the following) and $c(o) = 0$ for all other $o \in O$;

$P = C$;
T is the polytopic tile system described above;
$\mu(t_{t2}) = \{p_t\}$, $\mu(s_2) = \{p_y\}$, and $\mu(u_i) = \{p_i\}$, $\mu(\overline{u}_i) = \{\overline{p}_i\}$, $1 \le i \le n$, all proteins placed in centres of their respective tiles;
R contains several kinds of rules described below.

Metabolic rules:

$\{a_i \to a_{i+1},\ 1 \le i \le 5\} \cup$
$\{a_6 \to a_1,\ [p_t a_1 \to [p_t b_1,\ b_{m+1} \to yes,\ [p_y yes \to yes[p_y\} \cup$
$\{b_j[p_i \to b_{j+1}[p_i \mid C_j \text{ contains } x_i, 1 \le j \le m, 1 \le i \le n\} \cup$
$\{b_j[\overline{p}_i \to b_{j+1}[\overline{p}_i \mid C_j \text{ contains } \overline{x}_i, 1 \le j \le m, 1 \le i \le n\}$.

Creation rules:

$\{a_1 \to r,\ a_1 \to t_{t1},\ a_3 \to w,\ a_3 \to s_1,\ a_4 \to t_b,\ a_5 \to s_2,\ a_6 \to t_{t2}\} \cup$
$\{a_1 \to u_i \mid 2 \le i \le n\} \cup \{a_4 \to v_i,\ a_5 \to u_i,\ a5 \to \overline{u}_i \mid 1 \le i \le n\}$.

Destruction rules:

$\{a_2 r \to a_3,\ a_6 w \to a_1\} \cup \{a_6 v_i \to a_1 \mid 1 \le i \le n\}$.

$r = 2$;
$\sigma(g_i, g_j) = \emptyset$ for all $(g_i, g_j) \in \gamma$.

The initial configuration of the M system $\mathcal{M}$ contains the seed tile (rod) t_1 and a high concentration of objects a in the environment. The computation of the M system $\mathcal{M}_\Phi$ consist of a *generating phase* when all possible interpretations of the logical variables $x_1, \ldots, x_n$ are generated, and a *checking phase* during which the interpretations are checked in parallel whether they satisfy the formula.

5.1 Generating phase

During a computation of the M system the tiles U_i assemble to sequences $U_1, \ldots, U_n$. This process completes in n cycles, each consisting of six steps.

Lemma 1. *Let the space $\mathbb{R}^2$ where the M system $\mathcal{M}_\Phi$ is placed contains 2^k connected sequences of tiles $U_1, \ldots, U_k$, $1 \le k \le n-1$, covering all possible truth assignments to variables $x_1, \ldots, x_k$. Assume that all applicable creation/destruction rules of $\mathcal{M}_\Phi$ are always applied. Then, after 6 steps, there will be 2^{k+1} connected sequences $U_1, \ldots, U_{k+1}$, covering all possible truth assignments to variables $x_1, \ldots, x_{k+1}$.*

Proof. The six steps of computation proceed as follows:

1. All tiles U_i, $1 \le i \le k$, create and attach perpendicular rods r to connectors c_{in} and c_{out} (rule $a_1 \to r$) resulting in pushing of tiles in the direction of rods so that mutual distances between sequences are set to 5.
   Furthermore, each tile U_k attaches a new tile u_{k+1} to its connector c_{kt} (resp. $\overline{c}_{kt}$), lengthening the sequences $\{U_i\}$ to $k+1$ elements (rules $a_1 \to u_i$, $2 \le i \le n$).
2. Rods r are destroyed (rule $a_2 r \to a_3$).
3. Tiles w are attached to connectors $c_{(k+1)t}$ of tiles u_{k+1}, blocking further vertical growth in the next two steps (rule $a_3 \to w$).

4. Each U_i, $1 \leq i \leq k$, attaches to its connector c_{i1} (resp. $\bar{c}_{i1}$) tile v_i such that its longer edge is perpendicular to U_i (rules $a_4 \to v_i$, $1 \leq i \leq n$).
5. Each v_i, $1 \leq i \leq k$, attaches to its free connector a new tile $\overline{u}_i$ or u_i (rule $a_5 \to u_i$, $a_5 \to \overline{u}_i$, $1 \leq i \leq n$) such that configurations $u_i - v_i - \overline{u}_i$ or $\overline{u}_i - v_i - u_i$ are produced. Therefore, a "ladder" containing two complementary sequence $U_1, \ldots, U_k$ with v_i's as rungs is produce. The new tiles U_i also connect together by their facet connectors.
6. Tiles w and v_i, $1 \leq i \leq k$, are destroyed, disconnecting old sequences of tiles and new complementary sequences (rules $a_6 w \to a_1$, $a_6 v_i \to a_1$, $1 \leq i \leq n$). □

Lemma 2. *Assume that all applicable creation/destruction rules of $\mathcal{M}_\Phi$ are always applied. Then $\mathcal{M}_\Phi$ produces in $6n + 3$ initial steps of its computation a set of 2^n interconnected sequences of tiles $U_1 - \ldots - U_n$ covering all possible assignments to propositional variables $x_1, \ldots, x_n$. The sequences are enclosed in separate closed subspaces ("cells") composed of tiles $t_{t1}, s_1, t_b, s_2, t_{t2}$.*

Proof. The system $\mathcal{M}_\Phi$ starts with a single seed tile u_1 and with the environment containing many objects a_4. By Lemma 1, steps 4, 5, 6, also tile $\overline{u}_1$ is produced. Then, again by Lemma 1 and by the induction argument, sequences $U_1 - \ldots - U_n$ covering all possible assignments to $x_1, \ldots, x_n$ are produced in $6(n-1)$ steps. The computation proceeds as follows:

1. Perpendicular tiles t_{t1} are attached to by their connectors c_{t11} to tiles U_n ending the sequences (rules $a_1 \to t_{t1}$) so that t_{t1} are oriented towards the side *in* of U_n. Meantime, rods r are attached to all tiles U_i as in step 1 of Lemma 1, creating mutual space between sequences.
2. Rods r are destroyed.
3. Vertical tiles s_1 are attached to connectors c_{t11} of tiles t_{t1} (rules $a_3 \to s_1$) so that they are parallel to sides *in* of all tiles U_i in their associated sequence.
4. Horizontal tiles t_b are attached to connectors c_{1sb} of tiles s_1 (rules $a_4 \to t_b$), forming bottom of a future rectangular cell. Note that tiles v_i cannot be attached to $U_i's$ as in step 4 of Lemma 1 since their growth is blocked by s_1.
5. Vertical tiles s_2 are attached to connectors c_{tb1} of tiles t_b (rules $a_5 \to s_2$) so that they are parallel to sides *out* of all tiles U_i in their associated sequence.
6. Finally, tiles t_{t2} are attached to connectors c_{2st} of tiles s_2 (rules $a_6 \to t_{t2}$), enclosing the sequence $U_1, \ldots, U_n$ in a rectangular cell.

Observe that, after assembling the cells, no creation/destruction rules can be used anymore to alter their structure. □

5.2 Checking phase

In this phase the M system $\mathcal{M}_\Phi$ checks in parallel, whether any of the cells formed in the generating phase represent a model of the formula Φ.

Lemma 3. *The M system $\mathcal{M}_\Phi$ can release the object* yes *to the environment if and only if there is a sequence of tiles $U_1 - \ldots - U_n$ representing a model of Φ, enclosed in a "cell" composed of tiles $t_{t1}, s_1, t_b, s_2, t_{t2}$. In the affirmative case, the object yes appears in the environment in $\mathcal{O}(n.m^2)$ steps after forming the cell with probability $p > 3/4$.*

Proof. Note first that a sequence of rules leading eventually to production of the object *yes* must start with the rule $[p_t a_1 \to [p_t b_1$ using protein p_t. This protein appears only on tile t_{t2} which is assembled to the remaining tiles only as the last tile enclosing a cell composed of $t_{t1}, s_1, t_b, s_2, t_{t2}$. Hence, without forming a completed cell in the generating phase, no object yes can be produced.

If the cell is formed, the rule is applied with a probability close to 1 due to a high concentration of objects a_1 inside the cell, releasing objects b_1 inside. Then for each clause C_j, $1 \le j \le m$, with k literals, there are k rules of the form
$b_j[p_i \to b_{j+1}[p_i$ if C_j contains x_i,
$b_j[\overline{p}_i \to b_{j+1}[\overline{p}_i$ if C_j contains $\overline{x}_i$,
where proteins p_i are placed on tile u_i and $\overline{p}_i$ are placed on $\overline{u}_i$. Hence, object b_{j+1} can be produced if an only if the interpretation represented by tiles $U_1 - \ldots - U_n$ in the cell is a model of C_j. By induction, object b_{m+1} can be produced if and only if this interpretation is a model of all clauses $C_1, \ldots, C_m$, i.e., model of the whole formula Φ. Finally, by using the rules $b_{m+1} \to yes$ and $[p_y yes \to yes[p_y$, where protein p_y is placed on the tile s_2, object *yes* is eventually sent to the environment.

Assume now that the rules producing object *yes* can be applied, and calculate the probability of this event. Let us divide the "cell" into n vertically arranged rectangles, each containing tile U_i, $1 \le i \le n$. Each tile u_i ($\overline{u}_i$) has a protein p_i ($\overline{p}_i$) placed inj its center, so that the whole volume of i-th rectangle lies within its reaction radius $r = 2$. Therefore, whenever an object b_j enters i-th rectangle, a rule $b_j[P_i \to b_{j+1}[P_i$, where $P_i \in \{p_i, \overline{p}\}$, is applicable.

Let us calculate first the probability that a randomly moving object b_j visits i-th rectangle, for an arbitrary but fixed $1 \le i \le n$, in at least one of n consecutive steps:

$$P_1 = 1 - \left(\frac{n-1}{n}\right)^n > 1 - e^{-1} \text{ for each } n \ge 1,$$

since the value of the fraction converges from 0 to e^{-1} as $n \longrightarrow \infty$. Consequently, the probability that the object b_j visits the rectangle during $m \cdot n$ consecutive steps is greater than $1 - e^{-m}$. Hence, provided that the sequence $U_1 - \ldots - U_n$ represents a model of Φ, each of the rules $b_j[P_i \to b_{j+1}[P_i$, $1 \le j \le m$, is applied in the cell in $m \cdot n$ steps with probability $> 1 - e^{-m}$. The resulting probability of application of all these rules in nm^2 steps is $(1 - e^{-m})^m$. The rule $b_{m+1} \to yes$ is applied with certainty whenever the object b_{m+1} is produced, and the last rule $[p_y yes \to yes[p_y$ is again applied in $m \cdot n$ steps with probability $> 1 - e^{-m}$. Hence the final probability that the checking phase for a formula Φ with m clauses releases object *yes* to the environment in $nm(m+1)$ steps is

$$P_{\text{check}}(m) = (1 - e^{-m})^{m+1}.$$

Observe that $P_{\text{check}}(3) \approx 0.81 > 3/4$ and the value converges quickly to 1 with growing m. This concludes the proof.

□

Lemma 4. *The M system $\mathcal{M}_{\Phi}$ can send the object* yes *to the environment if and only if the formula Φ is satisfiable. Furthermore, in the affirmative case, the object yes appears in the environment in $\mathcal{O}(n.m^2)$ steps with probability $p > 1/2$.*

Proof. By lemma 3, the object *yes* is never produced in the M system $\mathcal{M}_{\Phi}$ if the formula Φ is unsatisfiable.

Consider now the case when Φ has a model corresponding to a connected sequence of tiles $U_1, \ldots, U_n$. The "cell" satisfying the assumption of Lemma 3 is assembled if none of the rules participating in construction of the sequence (creating or destroying tiles) failed due to absence of floating objects involved in the rule.

By definition, a rule is applied when all the objects at its left-hand side are present within the reaction radius r. Note that each rule of $\mathcal{M}_{\Phi}$ has a single floating object a_i at its left-hand side, $1 \le i \le 6$. by definition, the rule is applied when the floating object is located within a certain "reaction ball" of radius r with volume $v = \frac{4}{3}\pi r^3$. The ball is located in a (large) environment of a volume V, containing on average $Vc(a_i)$ objects. Assuming their uniform random distribution, the probability that the ball contains no object a_i is

$$P(\emptyset) = \Big(\frac{V-v}{V}\Big)^{Vc(a_i)} = \Big(\frac{zv-v}{zv}\Big)^{zvc(a_i)} = \Big[\Big(\frac{z-1}{z}\Big)^{z}\Big]^{vc(a_i)}$$

where we used substitution $z = V/v$. As the environment is large (unbounded), the expression in square brackets converges from bellow to e^{-1} with growing V. Observe furthermore that some of objects a_i in the ball can participate simple rules, too, hence only their fraction $\alpha \approx 1/2$ will be available for the chosen creation/destruction rule r_i. Therefore, the probability of application of a single instance of rule r_i using object a_i is

$$P(r) = 1 - P(\emptyset)^{\alpha} > 1 - e^{-v\alpha c(a_i)}.$$

To produce a randomly chosen sequence of tiles $U_1, \ldots, U_n$ enclosed in its "cell" by construction in the proof of Lemma 2 requires three rules applied in three initial steps, then cycles from $i = 1$ to $n-1$ such that $3i+4$ rules creating/destroying tiles $U_i/v_i/w$ are applied in each of them, and finally five steps to assemble the "cell." (We do not count rules creating/estroying perpendicular auxiliary rods as their failure is repaired in the next cycle.) Hence we get the total number

$$R(n) = 3 + 5 + \sum_{i=1}^{n-1}(3i+4) = \frac{3}{2}n(n-1) + 4(n-1) + 8 = \mathcal{O}(n^2)$$

of applications of creation/destruction rules. Therefore, it is enough to choose the concentration $c(a_1)$ such that

$$P(r)^{R(n)} > (1 - e^{-v\alpha c(a_i)})^{R(n)} \ge \frac{3}{4}$$

to guarantee that a "cell" satisfying the assumption of Lemma 3 is assembled. Given the values of the constants $v\alpha \approx \frac{16}{3}\pi \approx 17$, it is easy to verify that a reasonable concentration satisfies this condition.

Therefore, if the formula Φ is satisfiable, then a cell representing its model is assembled with probability $> \frac{3}{4}$ and by Lemma 3 the object *yes* is released to the environment with probability at least $\frac{3}{4} \cdot \frac{3}{4} > \frac{1}{2}$ which concludes the proof. $\square$

Corollary 1. $\mathbf{NP} \subseteq \mathbf{MRP}^{*}_{MR_1}$, *where* MR_1 *is the class of M systems with rules contains a single floating object at their left-hand side.*

Proof. It is easy to verify that the family of M systems $\{\mathcal{M}_{\Phi} \mid \Phi$ is a formula in 3CNF $\}$ defined above is polynomially uniform by Turing machines. Then the statement follows by Lemma 4. $\square$

6 Conclusions

We have further developed a recent new hybrid model, *M systems*, that leverages properties of self-assembly and P systems exhibit controlled growth and robustness akin to those observed in cell biology. The model is inspired by P systems and self-assembly and the new properties are obtained by introducing geometric concepts of shape and arrangement of atomic objects at specific locations. Basic abstract operations in the model include reactions among objects, their transport through protein channels, and their mutual interconnection, leading to construction and destruction of complex geometric structures, which are cell-inspired in the examples we have provided, but which can adopt virtually any geometric forms.

M systems has been proven to be computationally universal in the Turing sense [21], and in this paper we have studied their computational efficiency, showing that despite restrictions imposed by its geometry, the model is still capable, in a probabilistic way, to solve NP complete problems in polynomial time. As an added value, geometrical properties allow to identify some bottlenecks appearing in the simulated processes, as the concentration or kinetic bottlenecks. Note, e.g., that we define the result of computation by presence of a specific object(s) in the environment, but their concentration decreases rapidly as the size of the occupied part of the environment grows exponentially.

As some known models of P systems allow to solve even PSPACE-complete problems in polynomial time, a natural open question arises whether an analogous result can be obtained in the framework of M system, too.

We have further shown that M systems are universal in a perhaps more restricted but more biological sense, i.e., they exhibit a morphogenetic and homeostatic structure in their life cycle and can live forever by replication, unless environmental resources are consumed. We have also demonstrated their capability to grow complex cell-inspired information processing structures, providing a model of the cytoskeleton growth which in turn controls a process akin to biological mitosis.

We have also developed a software simulator of M systems to continue research on this models that is available at url `sosik.zam.slu.cz/msystem` or `bmc.memphis.edu/cytos`. Finally, we have begun to address several questions poised in by providing natural definitions of "injuries" and "self-repair." Many problems of interest arise. What kind of "injuries" will harm the model beyond repair? How exactly can injury be properly defined to establish more specific properties and limitations of self-healing? Third, adding evolutionary properties to the model is an intriguing possibility – the capability to evolve unenthropically towards more efficient behavior related to its specific goals, which can be of many kinds. To this end, the model should be equipped with a kind of abstract genetic code defining shapes of tiles and placement of connectors and other proteins on them. Perhaps the evolution of new floating objects and proteins and their mutual reactions should be allowed, too, reflecting the evolution of new "organic" molecules. This evolution may produce new development of models *in silico*, a kind of artificial life closer to biological life as we know it.

Acknowledgements

This work was supported by The Ministry of Education, Youth and Sports Of the Czech Republic 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/13/2016.

References

1. Banu-Demergian, I., Stefanescu, G.: The geometric membrane structure of finite interactive systems scenarios. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y. (eds.) 14th International Conference on Membrane Computing. pp. 63–80. Institute of Mathematics and Computer Science, Academy of Sciences of Moldova, Chisinau (2013)
2. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Pardini, G.: Spatial calculus of looping sequences. Theor. Comput. Sci. 412(43), 5976–6001 (2011)
3. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Pardini, G.: Simulation of spatial P system models. Theor. Comput. Sci. 529, 11–45 (2014)
4. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Pardini, G., Tesei, L.: Spatial P systems. Natural Computing 10(1), 3–16 (2011)
5. Bernardini, F., Brijder, R., Cavaliere, M., Franco, G., Hoogeboom, H.J., Rozenberg, G.: On aggregation in multiset-based self-assembly of graphs. Natural Computing 10(1), 17–38 (2011)
6. Bernardini, F., Brijder, R., Rozenberg, G., Zandron, C.: Multiset-based self-assembly of graphs. Fundamenta Informaticae 75(1-4), 49–75 (2007)
7. Bernardini, F., Gheorghe, M., Krasnogor, N., Giavitto, J.L.: On self-assembly in population p systems. In: Calude, C.S., Dinneen, M.J., Păun, G., Pérez-Jímenez, M.J., Rozenberg, G. (eds.) Unconventional Computation: 4th International Conference, UC 2005. pp. 46–57. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
8. Blount, P., S.I.Sukharev, P.C.Moe1, M.J.Schroeder, Guy, H., Kung, C.: Membrane topology and multimeric structure of a mechanosensitive channel protein of escherichia coli. The EMBO Journal 15(18), 4798–4805 (1996)

9. Bourgine, P., Lesne, A.: Morphogenesis: Origins of Patterns and Shapes. Springer complexity, Springer Berlin Heidelberg (2010)
10. Cardelli, L., Gardner, P.: Processes in space. In: Ferreira, F., Löwe, B., Mayordomo, E., Mendes Gomes, L. (eds.) Programs, Proofs, Processes: 6th Conference on Computability in Europe, CiE 2010. vol. 6158, pp. 78–87. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
11. Cavaliere, M., Mardare, R., Sedwards, S.: A multiset-based model of synchronizing agents: Computability and robustness. Theoretical Computer Science 391(3), 216 – 238 (2008)
12. Krasnogor, N., Gustafson, S., Pelta, D., Verdegay, J.: Systems Self-Assembly: Multidisciplinary Snapshots. Studies in Multidisciplinarity, Elsevier Science (2011)
13. Manca, V., Pardini, G.: Morphogenesis through moving membranes. Natural Computing 13(3), 403–419 (2014)
14. Pérez-Jiménez, M., Romero-Jiménez, A., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2, 265–285 (2003)
15. Păun, A., Popa, B.: P systems with proteins on membranes. Fundamenta Informaticae 72(4), 467–483 (2006)
16. Păun, A., Popa, B.: P systems with proteins on membranes and membrane division. In: Ibarra, O., Dang, Z. (eds.) DLT 2006. Lecture Notes in Computer Science, vol. 4036, pp. 292–303. Springer, Berlin (2006)
17. Păun, G.: Membrane Computing – An Introduction. Springer, Berlin (2002)
18. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
19. Robinson, K., Messerli, M.: Left/right, up/down: the role of endogenous electrical fields as directional signals in development, repair and invasion. Bioessays 25, 759766 (2003)
20. Schrödinger, E.: What Is Life? The Physical Aspect of the Living Cell. Trinity College, Dublin (1944)
21. Sosík, P., Smolka, V., Drastík, J., Moore, T., Garzon, M.: Morphogenetic and homeostatic self-assembled systems. In: Patitz, M.J., Stannett, M. (eds.) Unconventional Computation and Natural Computation: 16th Int. Conf., UCNC 2017. Lecture Notes in Computer Science, vol. 10240, pp. 144–159. Springer, Berlin (2017)
22. Tangirala, K., Caragea, D.: Generating features using burrows wheeler transformation for biological sequence classification. In: Pastor, O. et al. (ed.) Proceedings of the International Conference on Bioinformatics Models, Methods and Algorithms. pp. 196–203. SciTePress (2014)
23. Tomita, M.: Whole-cell simulation: a grand challenge of the 21st century. Trends Biotechnol. 19(6), 205–210 (2001)
24. Turing, A.: The chemical basis of morphogenesis. Philos. Trans. R. Soc. Lond. B 237, 7–72 (1950)
25. Watson, J., Crick, F.: A structure for deoxyribose nucleic acid. Nature 171, 737–738 (1953)
26. Winfree, E.: Models of Experimental Self-Assembly. Ph.D. thesis, Caltech (1998)
27. Winfree, E.: Self-healing tile sets. In: J. Chen, N. Jonoska, G.R. (ed.) Nanotechnology: Science and Computation, pp. 55–66. Natural Computing Series, Springer-Verlag (2006)
28. Maxwell–Boltzmann distribution, Wikipedia (cit 2017-1-29)
29. Ziegler, G.: Lectures on Polytopes. Graduate Texts in Mathematics, Springer-Verlag, New York (1995)

A M system modelling membrane formation, cytoskeleton growth and reproduction

The M system described in this section is inspired by the process of mitosis and cell replication, controlled by the growth of cytoskeleton, as illustrated schematically at Fig. 2. Observe that, even if this process relies on several relatively complex regulation mechanisms, the M system model can simulate it, at least at the morphological and dynamical level, with as few as 16 rules. All processes in the model are controlled by fully local interactions between its elements.

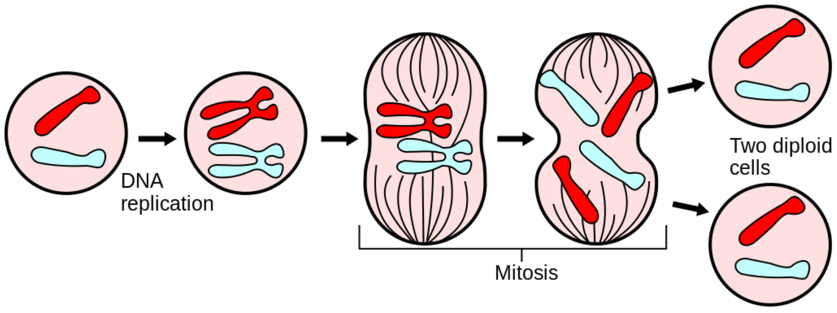


Fig. 2. Major events in mitosis. Licensed under Wikimedia Commons, authored by Mysid.

Let us start the formal description of the model by its polytopic tile system $T_0 = (Q, G, \gamma, d_g, S)$ in $\mathbb{R}^3$, which determines its spatial structure. Let

Q $= \{q_0, q_1, q_2, q_3, q_4, s_0, s_1, s_2\}$, where q_0, q_1, q_2 are larger pentagonal tiles forming cellular membrane, and q_3, q_4 are small pentagonal tiles forming nuclear membrane. All tiles contain facet connectors at all edges for mutual interconnection. There are three rods: s_0 and s_1 with two connectors at their endpoints, and s_2 is a rod with one connector at one endpoint and two fork-oriented connectors at the other endpoint. Rod s_0 is used only in the first step of the system. Rods s_1 and s_2 form straight and fork segments of cytoskeleton, respectively, and we call them *microtubules*. Furthermore,

- tile q_0 contains at the centre a point connector to which rod s_0 can connect by its endpoint connector;
- tile tile q_3 contains at the centre a point connector which can connect to the other end of rod s_0;
- tiles q_0 and q_1 contain at the centre a point connector to which rod s_1 can connect.

Let

Δ_p be a regular pentagon with distance 10 from center to vertices, with vertices denoted by $\mathbf{x}_1, \ldots, \mathbf{x}_5$;

$\varDelta'_p$ be a regular pentagon with distance 3 from center to vertices, with vertices denoted by $\mathbf{x}'_1, \ldots, \mathbf{x}'_5$);
$\varDelta_0 = \langle 0, 9.184\rangle$ be a segment with length 9.184;
$\varDelta_s = \langle 0, 1.4\rangle$ be a segment with length 1.4.

The tiles in Q are defined as follows:

$q_0 = (\varDelta_p,$ (pentagon)
$\{(\langle \mathbf{x}_i, \mathbf{x}_{i \bmod 5+1}\rangle, g_e, \varphi_p) \mid 1 \le i \le 5\}$ (facet connectors)
$\cup \{(\mathbf{0}, g_0, \pi/2), (\mathbf{0}, g_3, \pi/2)\},$ (point connectors)
$g_x),$ (surface glue)

$q_1 = (\varDelta_p,$
$\{(\langle \mathbf{x}_i, \mathbf{x}_{i \bmod 5+1}\rangle, g_a, \varphi_p) \mid 1 \le i \le 5\}$
$\cup \{(\mathbf{0}, g_3, \pi/2)\}, g_x),$

$q_2 = (\varDelta_p,$
$\{(\langle \mathbf{x}_1, \mathbf{x}_2\rangle, g_f, \varphi_p), (\langle \mathbf{x}_2, \mathbf{x}_3\rangle, g_b, \varphi_p), (\langle \mathbf{x}_3, \mathbf{x}_4\rangle, g_c, \varphi_p), (\langle \mathbf{x}_4, \mathbf{x}_5\rangle, g_d, \varphi_p),$
$(\langle \mathbf{x}_5, \mathbf{x}_1\rangle, g_b, \varphi_p)\}, g_x),$

$q_3 = (\varDelta'_p, \{(\langle \mathbf{x}'_i, \mathbf{x}'_{i \bmod 5+1}\rangle, g_a, \varphi_p) \mid 1 \le i \le 5\}$
$\cup \{(\mathbf{0}, g_2, -\pi/2),$
$g_t),$

$q_4 = (\varDelta'_p,$
$\{(\langle \mathbf{x}'_1, \mathbf{x}'_2\rangle, g_f, \varphi_p), (\langle \mathbf{x}'_2, \mathbf{x}'_3\rangle, g_b, \varphi_p), (\langle \mathbf{x}'_3, \mathbf{x}'_4\rangle, g_c, \varphi_p), (\langle \mathbf{x}'_4, \mathbf{x}'_5\rangle, g_d, \varphi_p),$
$(\langle \mathbf{x}'_5, \mathbf{x}'_1\rangle, g_b, \varphi_p)\}, g_x),$

$s_0 = (\varDelta_0, \{(\{0\}, g_1, \pi/2), (\{9.184\}, g_1, \pi/2)\}, g_x);$

$s_1 = (\varDelta_s, \{(\{0\}, g_3, 0), (\{1.4\}, g_4, 0)\}, g_x);$

$s_2 = (\varDelta_s, \{(\{0\}, g_5, 0), (\{1.4\}, g_6, \pi/30), (\{2\}, g_e, -\pi/30)\}, g_x);$

where $\varphi_p = 2.034443935795703$ rad is the inner angle between two faces of a dodecahedron.

$G = \{g_0, g_1, g_2, g_3, g_4, g_5, g_6, g_a, g_b, g_c, g_d, g_e, g_f, g_t, g_x\}$ is the set of glues;
$\gamma = \{(g_0, g_1), (g_1, g_2), (g_3, g_3), (g_4, g_5), (g_6, g_3), (g_4, g_t), (g_6, g_t), (g_a, g_f), (g_f, g_a),$
$(g_b, g_b), (g_c, g_c), (g_d, g_d), (g_e, g_f)\}$ is the glue relation;
$d_g = 0.1$ is the glue distance;
$S = \{q_0\}$ is the seed tile.

Then let us define the M system $\mathcal{M}_0 = (F, P, T_0, \mu, R, r, \sigma)$ such that:

$F = (O, m, c)$, where:
$O = \{a, b, c, x\}$ are floating objects;
$m(a) = 5$, $m(b) = 5$, $m(c) = 4$, $m(x) = 7$ is the mobility of floating objects;
$c(a) = 0.1$ and $c(o) = 0$ for all other $o \in O$ is the concentration of floating objects in the environment;
$P = \{p_0, p_1, p_2, p_3, p_4\}$;
T_0 is the polytopic tile system described above;

$\mu(q_2) = \{(p_0,(2,0)),(p_0,(2,2)),(p_2,(0,0)),(p_2,(0,2)),\} \cup \{(p_1,(i,j)) \mid i,j \in \{-5,5\}\}$, (proteins placed on tiles)
$\mu(q_4) = \{(p_3,(i,j)) \mid i,j \in \{-1,1\}\} \cup \{(p_4,(i,j)) \mid i,j \in \{0,1\}\}$,
$\mu(t) = \emptyset$ for all other $t \in Q$;
R contains the following rules:

Metabolic rules:
$a[p_0 \to [p_0 a$;
$[p_1 c \to [p_1 a$;
$[p_1 cx \to [p_1 aa$;
$[p_2 a \to [p_2 b$;
$[p_3 cx \to [p_3 aa$;
$cx[p_3 \to aa[p_3$;
$[p_4 a \to [p_4 cc$;

Creation rules:
$a^8 \to q_1$; (cellular membrane tiles creation from eight objects a)
$a^8 \to q_2$;
$aaa \to q_3$; (nuclear membrane tiles creation from three objects a)
$aaa \to q_4$;
$aaa \to s_0$; (auxiliary rod creation)
$b \to s_1$; (microtubules creation)
$b \to s_2$;

Division rules:
$g_c \overset{x}{—} g_c \to g_c, g_c$ (division of both cellular and nuclear membrane)
$g_d \overset{x}{—} g_d \to g_d, g_d$

$r = 14$ is the reaction distance;
$\sigma(g_4, g_t) = xxx$ and $\sigma(g,h) = \emptyset$ for all other $(g,h) \in \gamma$.

The M system $\mathcal{M}_0$ passes a (possibly infinite) sequence of configurations described below. Since the system is nondeterministic, the provided description captures its most probable development, with possible statistical deviations:

1. In the initial configuration there is a single seed tile q_0 for cellular membrane. There are objects a in the environment in concentration 0.1 per cubic unit.
2. In the first step, the auxiliary rod s_0 is created and attached to the centre of the seed tile q_0 so that it is perpendicular to it. Simultaneously, five cellular membrane tiles q_2 are created by the rule $a^8 \to q_2$, and attached to five edge connectors of the tile q_0. Their other connectors at mutually matching positions with glues g_b attach together, too.
3. In the second step, tile q_3 is created and attached perpendicularly to rod s_0, hence it is parallel with q_0. Simultaneously, five more tiles q_2 are connected to the existing tiles q_2, forming subsequently the closed structure playing the role of cellular membrane.

4. In the third step, pentagonal tile q_1 concludes the cellular membrane building phase, completing the dodecahedron-shaped cell ("soccer ball") and enclosing inside rather large number of objects a (over a hundred of thousands). Simultaneously, an analogous process of membrane creation on a smaller scale starts from tile q_3 to which five tiles q_4 are attached, and the nuclear membrane formation is completed in next two steps.
5. During the previous two steps, the reaction proteins p_2 on tile q_2 already started to catalyze the reaction of objects a to b applying the rule $p_2a \to p_2b$. As each tile q_2 contains two proteins, and the cellular membrane contains 10 tiles q_2, 20 objects b is produced at each step within the cell. An analogous process runs in the nuclear membrane with proteins on tiles q_4 and objects c.
6. The objects b allow for creation of the microtubules s_1 and s_2, which can connect to point connectors in centres of "polar" tiles q_0 and q_1 of the cellular membrane. A non-deterministic growth of cytoskeleton towards the interior of the cell starts, consuming objects b as it continues.
7. When the microtubules s_1 and s_2 of the growing cytoskeleton reach the nuclear membrane, they can attach to its polar tiles q_3, each such contact releasing three objects x (as defined in the mapping σ).
8. Each object x can break one bond of a pair of tiles q_2 (or q_4) in the equatorial part of both membranes.
9. When all these bonds (10 in each membrane) are broken, the cell splits into two parts, each containing a half of the outer membrane, a part of the cytoskeleton growing from it, and the corresponding half of the nucleus.
10. The same growth rules as in the second and the third step now allow to complete both halves into two complete cells, pushing them apart as they grow. Before concluding this process, however, two important events take place: (i) both open halves of the cellular membrane are replenished with objects a from the environment, and (ii) open nucleus releases a large amount of accumulated objects c, while replenishing a's. Objects c diffuse into both halves of the cell, deactivating objects x via the rules $[p_1cx \to [p_1aa$, $[p_3cx \to [p_3aa$ and $cx[p_3 \to aa[p_3$.
11. After two steps during which the two new cells are completed, cytoskeleton growth can continue and the process of the "cell" division is repeated.

In such a way, the cells in the M system continue their division and the whole system is in a dynamical homeostasis: if environmental resources (represented by objects a) are unlimited, then so is the growth of the system. In the limited environment the growth would stop after exhaustion of the environmental resources.

Notice that, as the model is nondeterministic, the mitosis can be sometimes blocked, for instance because the objects x regulating the division process may start another nucleus division before the whole cell is ready to divide. This synchronization disorder in division control makes the cell dysfunctional and incapable of further mitosis.

Counting Membrane Systems

Luis Valencia-Cabrera, David Orellana-Martín,
Agustín Riscos-Núñez, 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: {lvalencia, dorellana, ariscosn, marper}@us.es

Abstract. A decision problem is one that has a yes/no answer, while a counting problem asks how many possible solutions exist associated with each instance. Every decision problem X has associated a counting problem, denoted by $\#X$, in a natural way by replacing the question "*is there a solution*?" with "*how many solutions are there*?". Counting problems are very attractive from a computational complexity point of view: if X is an **NP**-complete problem then the counting version $\#X$ is **NP**-hard, but the counting version of some problems in class **P** can also be **NP**-hard.
In this paper, a new class of membrane systems is presented in order to provide a natural framework to solve counting problems. The class is inspired by a special kind of non-deterministic Turing machines, called counting Turing machines, introduced by L. Valiant. A polynomial-time and uniform solution to the counting version of the SAT problem (a well-known #**P**-complete problem) is also provided, by using a family of counting polarizationless P systems with active membranes, without dissolution rules and division rules for non-elementary membranes but where only very restrictive cooperation (minimal cooperation and minimal production) in object evolution rules is allowed.

Key words: Membrane Computing, polarizationless P systems with active membranes, cooperative rules, the **P** versus **NP** problem, #SAT problem.

1 Introduction

Membrane Computing is a computational paradigm inspired by the structure and functioning of the living cells as well as from the cooperation of cells in tissues, organs, and organisms. This paradigm provides distributed parallel and non-deterministic computing models. All of them share the main syntactical ingredients: a finite alphabet (the *working* alphabet whose elements are called *objects*), a finite set of processor units delimiting *compartments* (called *membranes*, *cells* or *neurons*) interconnected by a *graph-structure* in such manner that initially each processor contains a multiset of objects, a finite set of *evolution rules* which provides the dynamic of the system, and an *environment*.

According to the inspiration they come from, there are basically three approaches: *cell-like* P systems, where the compartments are arranged like in a living cell (that is, in a hierarchical structure) [7]; *tissue-like* P systems, whose inspiration comes from the living tissues, where cells bump into each other and communicate through pores or other membrane mechanisms [20]; and *neural-like* P systems, which mimic the way that neurons communicate with each other by means of short electrical impulses, identical in shape (voltage), but emitted at precise moments of time [10]. The term *membrane system* is used to refer to *cell-like* P systems, *tissue-like* P systems or *neural-like* P systems indistinctly.

In cell-like and neural-like P systems the environment plays a "passive" role in the sense that it only receives objects, but cannot contribute objects to the system. However, in tissue-like P systems the role played by the environment is "active"in the sense that it can receive objects from the system and also send objects inside the system, and the objects initially placed in the environment have an arbitrarily large number of copies.

Decision problems (those having a `yes`/`no` answer) have associated a language in a natural way, in such manner that solving a decision problem is expressed in terms of the recognition of the language associated with it. In this context, recognizer membrane systems was introduced in order to define what solving a decision problem means in the framework of Membrane Computing [12]. P systems with active membranes was introduced in [8, 9]. These cell-like models make use of electrical charges associated with membranes and division rules. They have the ability to provide efficient solutions to computationally hard problems, by making use of an exponential workspace created in a polynomial time. The class of decision problems which can be solved by families of P systems with active membranes with dissolution rules and which use division for elementary and non-elementary membranes is equal to **PSPACE** [14]. However, if electrical charges are removed from the usual framework of P systems with active membranes, then dissolution rules come to play a relevant role (without them, only problems in class **P** can be solved in an efficient way, even in the case that division for non-elementary membranes are permitted [5]). P systems with active membranes and without polarizations were initially studied in [1, 2] by replacing electrical charges by the ability to change the label of the membranes.

Counting problems (those asking how many possible solutions exist associated with each instance) have a natural number, instead of a `yes`/`no`, as an answer. Each decision problem has associated a counting problem in a natural way by replacing "*there exists a solution*" with "*how many solutions*". For instance, the counting problem associated with the `SAT` problem (denoted by #`SAT`) is the following: given a Boolean formula φ in conjunctive normal form, how many truth assignments make true φ? It is worth pointing out that the counting problem associated with a decision problem may be harder than the decision problem, from a complexity point of view.

The main goal of this paper is twofold. On the one hand, to provide a formal framework in Membrane Computing to solve counting problems by introducing *counting membrane systems*. This approach was first initiated by A. Alhazov et al. [3] and now, the computing model is formally defined inspired by *counting Turing machines* introduced by L. Valiant in 1979: "*a standard nondeterministic TM with an auxiliary output device that (magically) prints in binary notation on a special tape the number of accepting computations induced by the input*". L. Valiant also introduced the complexity class #**P** of functions that can be computed by *counting Turing machines* running in polynomial time [19]. The concept of #**P**-complete problems is defined in a natural way by considering *parsimonious reduction*, that is reduction which preserves the number of solutions.

On the other hand, following the works initiated in [15–17], a uniform and polynomial-time solution to the #`SAT` problem, a well-known #**P**-complete problem, is provided by means of a family of counting membrane systems from $\mathcal{DAM}^0_{\mathtt{c}}(mcmp, +c, -d, -n)$ whose elements are (counting) polarizationless P systems with active membranes where labels of membranes keep unchanged by the application of rules, but where dissolution rules and division rules for non-elementary membranes are forbidden and some kind of very restrictive cooperation in object evolution rules is allowed.

The paper is structured as follows. Next, we shortly recall some preliminary basic definitions related to abstract problems. Section 3 introduces counting membrane systems, and the concept of uniform polynomial-time solvability of counting problems by means of families of counting membrane systems is presented (specifically, the class $\mathcal{DAM}^0_{\mathtt{c}}(mcmp, +c, -d, -n)$ is defined). Section 4 is devoted to showing a uniform and polynomial-time solution of the #`SAT` by using a family of counting polarizationless P systems with active membranes, without dissolution rules and with division only for elementary membranes where minimal cooperation and minimal production is allowed in object evolution rules. The paper ends with some conclusions and final remarks.

2 Abstract problems

Roughly speaking, an *abstract problem* is a "general question to be answered, usually possessing several parameters whose values are left unspecified" [4]. Solving an abstract problem consists of answering the question associated with it. Thus, an abstract problem consists of a (finite or infinite) set of *concrete problems*, called *instances*, obtained by specifying particular values for all parameters. Each instance has an associated set (eventually empty) of possible *solutions* and the answer to the general question of the problem is related to that set.

A *search problem* (or *function problem*) is an abstract problem such that the question is to identify/find *one* solution from the set of possible solutions associated with each instance. For example, given a Boolean formula φ in conjunctive normal form to find any truth assignment which makes it true, or if there is no such truth assignment, answer "`no`". That is, in this problem a "function" must be computed in such manner that for every input formula φ, this "function" may have many possible outcomes (any satisfying truth assignments) or none.

A *decision problem* is a particular case of search problem. Specifically, a decision problem can be viewed as an abstract problem that has a `yes` or `no` answer. This kind of problems can be formulated by specifying a generic instance of the problem and by stating a `yes`/`no` question concerning to the generic instance [4]. For example, the `SAT` problem is the following decision problem: given a Boolean formula in conjunctive normal form, is there a truth assignment that makes the formula true?

Informally, a *counting problem* is an abstract problem such that one asks how many possible solutions exist associated with each generic instance, that is, in this kind of problems the output is a natural number rather than just `yes` or `no` as in a decision problem. For example, the #`SAT` problem previously defined is a particular case of a counting problem.

In *optimization problems* we seek to find a *best solution* associated with each instance among a collection of feasible solutions, according to a concept of optimality given by an objective function associated with the problem. For example, the `MAXSAT` problem is the following optimization problem: given a Boolean formula in conjunctive normal form and a natural number k, is there a truth assignment that makes true at least k of the clauses?

Next, we formally define the previous concepts. A *search problem* (or *function problem*) X is a tuple (Σ_X, I_X, S_X) such that: (a) Σ_X is a finite alphabet; (b) I_X is a language over Σ_X whose elements are called *instances* of X; and (c) S_X is a function whose domain is I_X and for each $u \in I_X$, $S_X(u)$ is a set whose elements are called *solutions* for u. To *solve a search problem* X means the following: for each instance $u \in I_X$ return one element of $S_X(u)$ in the case that $S_X(u) \neq \emptyset$; otherwise, return "`no`". Each search problem $X = (\Sigma_X, I_X, S_X)$ has an associated binary relation Q_X defined as follows: $Q_X = \{(u, z) \mid u \in I_X \wedge z \in S_X(u)\}$. Then, solving the search problem X can be interpreted as follows: given an instance $u \in X$, find one element z such that $(u, z) \in Q_X$. We say that a deterministic Turing machine M solves a search problem X if, given as input any instance $u \in I_X$, the machine M with input u returns some element belonging to $S_X(u)$ (M accepts u) in the case that $S_X(u) \neq \emptyset$; otherwise, it returns "`no`" (M rejects u). That is, the Turing machine M computes a multivalued function F on I_X: this function may have many possible outcomes or none.

An *optimization problem* X is a tuple $(\Sigma_X, I_X, S_X, O_X)$ such that:

– (Σ_X, I_X, S_X) is a search problem.
– O_X is a function whose domain is I_X and for each instance $u \in I_X$ and for each possible solution $a \in S_X(u)$ associated with u, $O_X(u, a)$ is a positive rational number.
– For each instance $u \in I_X$ there exists a solution $a \in S_X(u)$ such that either $\forall b\,(b \in S_X(u) \Rightarrow O_X(u, b) \leq O_X(u, a))$ (we say that a is a *maximal solution* to instance u), or $\forall b\,(b \in S_X(u) \Rightarrow O_X(u, b) \geq O_X(u, a))$ (we say that a is a *minimal solution* to instance u).

To *solve an optimization problem* X means the following: for each instance $u \in I_X$, return a *maximal solution* or a *minimal solution.* We say that a deterministic Turing machine M solves an optimization problem X if, given an instance $u \in I_X$, the machine M with input u returns one *optimal* (maximal or minimal) solution associated with that instance.

A *decision problem* X is a search problem (Σ_X, I_X, S_X) such that for each instance $u \in I_X$, $S_X(u) = \{0\}$ or $S_X(u) = \{1\}$. In the case $S_X(u) = \{0\}$ we say that the answer of the decision problem is *negative* (`no`) for instance u. In the case $S_X(u) = \{1\}$ we say that the answer of the decision problem is *affirmative* (`yes`) for instance u. To *solve a decision problem* X means the following: for each instance $u \in I_X$, return `yes` in the case $S_X(u) = \{1\}$, otherwise, return `no`. Let us notice that a decision problem $X = (\Sigma_X, I_X, S_X)$ can be viewed as an optimization problem $(\Sigma_X, I_X, S_X, O_X)$ where $O_X(u, a)$ is constant, always equal to 1 (recall that for each instance $u \in I_X$ the set of possible solutions $S_X(u)$ is a singleton, either $\{0\}$ or $\{1\}$). Each decision problem $X = (\Sigma_X, I_X, S_X)$ has an associated language L_X defined as follows: $L_X = \{u \in \Sigma_X^* \mid S_X(u) = \{1\}\}$. Conversely, each language L over an alphabet Γ has an associated decision problem $X_L = (\Sigma_{X_L}, I_{X_L}, S_{X_L})$ defined as follows: $\Sigma_{X_L} = \Gamma$, $I_{X_L} = \Gamma^*$ and $S_{X_L}(u) = \{1\}$, for each $u \in L$, and $S_{X_L}(u) = \{0\}$, for each $u \notin L$. According to these definitions, for each decision problem X we have $X_{L_X} = X$ and for each language L we have $L_{X_L} = L$. A deterministic Turing machine M is said to solve a decision problem X if machine M *recognizes* or *decides* the language L_X associated with the problem X, that is, for any string u over Σ_X, if $u \in L_X$, then the answer of M on input u is *yes* (that is, M accepts u), and the answer is *no* otherwise (that is, M rejects u). A *non-deterministic* Turing machine M is said to solve a decision problem X if machine M *recognizes* L_X, that is, for any string u over Σ_X, $u \in L_X$ if and only if there exists $\underline{\text{at least one}}$ computation of M with input u such that the answer is *yes*.

A *counting problem* X is a tuple $(\Sigma_X, I_X, S_X, F_X)$ such that (Σ_X, I_X, S_X) is a search problem and F_X is the function whose domain is I_X, defined as follows: $F_X(u) = |S_X(u)|$, where $|S_X(u)|$ denotes the number of elements of the set $S_X(u)$, for each instance $u \in I_X$. A counting problem X can be considered as a particular case of a search problem expressed as follows: given an instance $u \in I_X$, how many z are there such that $(u, z) \in Q_X$? (where Q_X is the binary relation associated with the search problem). A counting Turing machine M is said to solve a counting problem X if, given an instance $u \in I_X$, the number of the accepting computations of M with input u is equal to the number of elements of the set $S_X(u)$, that is, the number of possible solutions associated with u.

3 Counting membrane systems

The main purpose of computational complexity theory is to provide bounds on the amount of computational resources necessary for any mechanical procedure that solves an abstract problem. Usually, this theory deals with languages encoding/representing decision problems. The solvability of decision problems is expressed in terms of recognize/decide the languages associated with them. In order to formally define what it means to solve decision problems in Membrane

Computing, a new variant called *recognizer membrane systems* was introduced in [12] (so-called *accepting* P systems) for cell-like P systems, in [11] for tissue-like P systems, and in [6] for neural-like P systems (so-called *accepting* spiking neural P systems). Next, a new class of membrane systems, called *counting membrane systems*, is introduced as a framework where counting problems can be solved in a natural way. These systems are inspired from *counting Turing machines* introduced by L. Valiant [19] and from recognizer membrane systems where the Boolean answer of these systems is replaced by an answer encoded by a natural number expressed in a binary notation (placed in the environment associated with the halting configuration).

Definition 1. *A counting membrane system Π is a membrane system such that:*

- *There exist two distinguished disjoint alphabets Σ (input alphabet) and Φ (final alphabet) both of them strictly contained in the working alphabet Γ of Π. Furthermore, a total order in the final alphabet $\Phi = \{a_0, a_1, \dots, a_n\}$ is considered.*
- *The membrane system has an input compartment labelled by i_{in}.*
- *All computations of the system halt.*
- *For each computation of the system, the environment associated with the corresponding halting configuration, may contain objects from Φ, but each of them with multiplicity at most one.*

According to Definition 1, the *result of any computation $\mathcal{C}$* of a counting P system is a natural number whose binary expression is encoded by the objects from the final alphabet placed in the environment associated with its halting configuration, according to the following criterion: (a) if the set of objects in Φ placed in the environment of the corresponding halting configuration is $\{a_{i_1}, \dots, a_{i_r}\}$, then the answer of $\mathcal{C}$ is the natural number $2^{i_1} + \ldots + 2^{i_r}$; and (b) if that set is the empty set then the answer of $\mathcal{C}$ is 0.

Many different classes of counting membrane systems depending on the kind of rules can be considered. For example, transition counting P systems, polarizationless counting P systems with active membranes, tissue counting P systems with symport/antiport rules can be defined in a natural way. Then, we will use a subscript **c** to emphasize that we are dealing with some kind of counting membrane system. For instance, $\mathcal{DAM}^0_{\mathbf{c}}(+e, +c, -d, -n)$ denotes the class of all polarizationless counting P systems with active membranes which use object evolution rules, communication rules and division rules only for elementary membranes, but dissolution rules are not allowed.

It is worth pointing out that any recognizer membrane system Π can be considered as a "particular case" of counting membrane system, where the final alphabet Φ is a singleton alphabet $\{a_0\}$ and the rules of the counting system are obtained from the rules of the recognizer system replacing `yes` with a_0 and replacing object `no` with a garbage object $\natural$ different from a_0.

3.1 Polynomial complexity classes for counting membrane systems

The concept of polynomial encoding in recognizer membrane systems was introduced in [13] and polynomial encodings are stable under polynomial-time reductions. This concept can be translated to counting membrane systems in a natural way.

Definition 2. *Let X be a counting problem whose set of instances is I_X. Let $\mathbf{\Pi} = \{\Pi(n) : n \in \mathbb{N}\}$ be a family of counting membrane systems. A* polynomial encoding *of X in $\mathbf{\Pi}$ is a pair (cod, s) of polynomial-time computable functions over I_X such that $s(u)$ is a natural number (obtained by means of a* reasonable *encoding scheme) and $cod(u)$ is a multiset over the input alphabet of $\Pi(s(u))$, for each instance $u \in I_X$.*

Definition 3. *A counting problem $X = (\Sigma_X, I_X, S_X, F_X)$ is* solvable in polynomial time and in a uniform way *by a family of counting membrane systems $\mathbf{\Pi} = \{\Pi(n) : n \in \mathbb{N}\}$ from a class $\mathcal{R}_c$, denoted by $X \in \mathbf{PCMS}_{\mathcal{R}_c}$, if the following holds:*

- *The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(n)$ from $n \in \mathbb{N}$ (in unary).*
- *There exists a polynomial encoding (cod, s) of X in $\mathbf{\Pi}$ such that:*
  - *The family $\mathbf{\Pi}$ is polynomially bounded with respect to (X, cod, s), that is, there exists a natural number $k \in \mathbb{N}$ such that for each instance $u \in I_X$, every computation of the system $\Pi(s(u))$ with input $cod(u)$ performs at most $|u|^k$ steps.*
  - *For each instance $u \in I_X$ and for each computation $\mathcal{C}$ of $\Pi(s(u))$ with input $cod(u)$ we have the result of $\mathcal{C}$ is $F_X(u)$.*

Having in mind that any recognizer membrane system Π can be considered as a "particular case" of counting membrane system, we have $\mathbf{PMC}_{\mathcal{R}} \subseteq \mathbf{PCMS}_{\mathcal{R}}$, for any class of recognizer systems $\mathcal{R}$.

3.2 Counting membrane systems from $\mathcal{DAM}^0_c(mcmp, +c, -d, -n)$

Let us recall that $\mathcal{DAM}^0(+e, +c, -d, -n)$ denotes the class of all recognizer polarizationless P systems with active membranes (μ denotes the membrane structure, Γ denotes the working alphabet and H denotes the set of labels) such that the set of rules is of the following forms:

⋆ $[\, a \rightarrow u \,]_h$ for $h \in H$, $a \in \Gamma$, u is a finite multiset over Γ (*object evolution rules*).

⋆ $a\,[\;\;]_h \rightarrow [\, b \,]_h$ for $h \in H$, $a, b \in \Gamma$ and h is not the label of the root of μ (*send-in communication rules*).

⋆ $[\, a \,]_h \rightarrow b\,[\;\;]_h$ for $h \in H$, $a, b \in \Gamma$ (*send-out communication rules*).

- ⋆ $[\,a\,]_h \to [\,b\,]_h\ [\,c\,]_h$ for $h \in H$, $a, b, c \in \Gamma$ and h is the label of an elementary membrane different of the root of μ (*division rules for elementary membranes*).

It is well known [5] that only problems in class **P** can be solved in polynomial time (and in a uniform way) by means of families from $\mathcal{DAM}^0(+e, +c, -d, -n)$. Moreover, this holds even in the case that division rules for elementary and non-elementary membranes are permitted.

By incorporating a restricted cooperation in object evolution rules, a uniform polynomial-time solution to the `SAT` problem, a well-known **NP**-complete problem [4], has been provided [17]. Specifically, *minimal cooperation and minimal production* (**mcmp**) in object evolution rules has been considered, that is, rules of the forms $[\,a \to b\,]_h$ or $[\,a\,b \to c\,]_h$, where $a, b, c \in \Gamma$, but at least one object evolution rule is of the second type. The corresponding class of recognizer P systems was denoted by $\mathcal{DAM}^0(mcmp, +c, -d, -n)$. Then we denote by $\mathcal{DAM}^0_{\mathbf{c}}(mcmp, +c, -d, -n)$ the class of all counting polarizationless P systems with active membranes, with minimal cooperation and minimal production in object evolution rules, with communication rules and division rules only for elementary membranes, but without dissolution rules.

4 A solution to #`SAT` in $\mathcal{DAM}^0_{\mathbf{c}}(mcmp, +c, -d, -n)$

In this section a uniform and polynomial-time solution to the counting problem #`SAT` problem, a well-known #**P**-complete problem, is provided by means of a family of counting membrane systems from $\mathcal{DAM}^0_{\mathbf{c}}(mcmp, +c, -d, -n)$. For that, the solution to the `SAT` problem given in [17] by using a family of membrane systems from $\mathcal{DAM}^0(mcmp, +c, -d, -n)$ is adapted, basically, in what concerns to the output stage.

Let us recall that the polynomial-time computable function (the *Cantor pair function*) $\langle n, p\rangle = ((n+p)(n+p+1)/2) + n$ is a primitive recursive and bijective function from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$. The family $\mathbf{\Pi} = \{\Pi(t) \mid\ t \in \mathbb{N}\}$ is defined in such a manner that system $\Pi(t)$ will process any Boolean formula φ in conjunctive normal form (CNF) with n variables and p clauses, where $t = \langle n, p\rangle$, provided that the appropriate input multiset $cod(\varphi)$ is supplied to the system (through the corresponding input membrane), and will answer how many truth assignments make true the input formula φ.

For each $n, p \in \mathbb{N}$, we consider the recognizer counting P system

$$\Pi(\langle n, p\rangle) = (\Gamma, \Sigma, \Phi, H, \mu, \mathcal{M}_1, \mathcal{M}_2, \mathcal{R}, i_{in})$$

from $\mathcal{DAM}^0(mcmp, +c, -d, -n)$, defined as follows:

(1) Working alphabet $\Gamma = \{\beta\,,\, \natural\} \cup \{\alpha_i \mid 0 \le i \le 2n + 2p + 1\} \cup$
$\{a_{i,j}\,, \mid 0 \le i \le n-1,\, 0 \le j \le i\} \cup \{a_i\,,\, \gamma_i \mid 0 \le i \le n-1\} \cup$
$\{b_{i,k} \mid 1 \le i \le n, 1 \le k \le i\} \cup \{c_j \mid 1 \le j \le p\} \cup$
$\{d_j \mid 2 \le j \le p\} \cup \{t_{i,k}, f_{i,k} \mid 1 \le i \le n, i \le k \le n + p - 1\} \cup$
$\{T_{i,k}, F_{i,k} \mid 1 \le i \le n, 0 \le k \le n-1\} \cup \{T_i, F_i \mid 1 \le i \le n\} \cup$
$\{x_{i,j,k}, \overline{x}_{i,j,k}, x^*_{i,j,k} \mid 1 \le i \le n, 1 \le j \le p, 0 \le k \le n + p\}$.

(2) Input alphabet $\varSigma = \{x_{i,j,0}, \overline{x}_{i,j,0}, x^*_{i,j,0} \mid 1 \le i \le n, 1 \le j \le p\}$.
(3) Final alphabet $\varPhi = \{a_i \mid 0 \le i \le n-1\}$.
(4) $H = \{1,2\}$.
(5) Membrane structure: $\mu = [\ [\ \ ]_2\ ]_1$, that is, $\mu = (V, E)$ where $V = \{1,2\}$ and $E = \{(1,2)\}$.
(6) Initial multisets: $\mathcal{M}_1 = \{\alpha_0^n\}$, $\mathcal{M}_2 = \{\beta, b_{i,1}, T^p_{i,0}, F^p_{i,0} \mid 1 \le i \le n\}$.
(7) The set of rules $\mathcal{R}$ consists of the following rules:

7.1 Rules for a general counter.

$[\ \alpha_k \longrightarrow \alpha_{k+1}\ ]_1$, for $0 \le k \le 2n+2p$

7.2 Rules to generate all truth assignments.

$[\ b_{i,i}\ ]_2 \longrightarrow [\ t_{i,i}\ ]_2\ [\ f_{i,i}\ ]_2$, for $1 \le i \le n$
$[b_{i,k} \longrightarrow b_{i,k+1}\ ]_2$, for $2 \le i \le n \wedge 1 \le k \le i-1$

7.3 Rules to generate suitable objects in order to start the next stage.

$$\left.\begin{array}{l}[t_{i,k} \longrightarrow t_{i,k+1}\ ]_2\\ [f_{i,k} \longrightarrow f_{i,k+1}\ ]_2\end{array}\right\} 1 \le i \le n-1 \wedge i \le k \le n-1$$
$$\left.\begin{array}{l}[T_{i,k} \longrightarrow T_{i,k+1}\ ]_2\\ [F_{i,k} \longrightarrow F_{i,k+1}\ ]_2\end{array}\right\} 1 \le i \le n, 0 \le k \le n-2$$
$$\left.\begin{array}{l}[T_{i,n-1} \longrightarrow T_i\ ]_2\\ [F_{i,n-1} \longrightarrow F_i\ ]_2\end{array}\right\} 1 \le i \le n$$

7.4 Rules to filter out either T_i or F_i, according to each truth assignment.

$$\left.\begin{array}{l}[t_{i,k}\ F_i \longrightarrow t_{i,k+1}\ ]_2\\ [f_{i,k}\ T_i \longrightarrow f_{i,k+1}\ ]_2\end{array}\right\} 1 \le i \le n \wedge n \le k \le n+p-2$$
$$\left.\begin{array}{l}[t_{i,n+p-1}\ F_i \longrightarrow \natural\ ]_2\\ [f_{i,n+p-1}\ T_i \longrightarrow \natural\ ]_2\end{array}\right\} 1 \le i \le n$$

7.5 Rules to prepare the input formula for check clauses.

$$\left.\begin{array}{l}[\ x_{i,j,k} \longrightarrow x_{i,j,k+1}\ ]_2\\ [\ \overline{x}_{i,j,k} \longrightarrow \overline{x}_{i,j,k+1}\ ]_2\\ [\ x^*_{i,j,k} \longrightarrow x^*_{i,j,k+1}\ ]_2\end{array}\right\} 1 \le i \le n,\ 1 \le j \le p,\ 0 \le k \le n+p-1$$

7.6 Rules for the first checking stage.

$$\left.\begin{array}{l}[T_i\ x_{i,j,n+p} \longrightarrow c_j\ ]_2\\ [T_i\ \overline{x}_{i,j,n+p} \longrightarrow \natural\ ]_2\\ [T_i\ x^*_{i,j,n+p} \longrightarrow \natural\ ]_2\\ [F_i\ x_{i,j,n+p} \longrightarrow \natural\ ]_2\\ [F_i\ \overline{x}_{i,j,n+p} \longrightarrow c_j\ ]_2\\ [F_i\ x^*_{i,j,n+p} \longrightarrow \natural\ ]_2\end{array}\right\} 1 \le i \le n \wedge 1 \le j \le p$$

7.7 Rules for the second checking stage.

$[c_1\ c_2 \longrightarrow d_2\ ]_2$
$[d_j\ c_{j+1} \longrightarrow d_{j+1}\ ]_2$, for $2 \le j \le p-1$

7.8 Rules to prepare objects in the skin membrane.

$[\ \beta\ d_p \longrightarrow \gamma_0\ ]_2$
$[\ \gamma_0\ ]_2 \longrightarrow \gamma_0\ [\ ]_2$, for $0 \le i \le n-1$

7.9 Rules to prepare objects encoding the binary output.

$[\ \gamma_i^2 \longrightarrow \gamma_{i+1}\ ]_1$, for $0 \le i \le n-2$
$[\ \alpha_{2n+2p+1}\ \gamma_i \longrightarrow a_{i,0}\ ]_1$, for $0 \le i \le n-1$
$[\ a_{i,j} \longrightarrow a_{i,j+1}\ ]_1$, for $1 \le i \le n-1, 0 \le j \le i-1$

7.10 Rules to produce the output.
$[\ a_{i,i}\ ]_1 \longrightarrow\ a_i\ [\ ]_1\,,$ for $0 \le i \le n-1$

(8) The input membrane is the membrane labelled by 2 ($i_{in} = 2$) and the output region is the environment.

4.1 An overview of the computation

It is easy to check that each P system $\Pi(\langle n,p\rangle)$ previously defined is deterministic.

We consider the polynomial encoding (cod, s) from #SAT in $\mathbf{\Pi}$ defined as follows: let φ be a Boolean formula in conjunctive normal form. Let $Var(\varphi) = \{x_1, \cdots, x_n\}$ be the set of propositional variables and $\{C_1, \cdots, C_p\}$ the set of clauses of φ. Let us assume that the number of variables and the number of clauses of the input formula φ, are greater than or equal to 2. Then, we define $s(\varphi) = \langle n,p\rangle$ and

$$cod(\varphi) = \{x_{i,j,0} \mid\ x_i \in C_j\} \ \cup\ \{\overline{x}_{i,j,0} \mid \neg x_i \in C_j\} \ \cup\ \{x^*_{i,j,0} \mid x_i \notin C_j, \neg x_i \notin C_j\}$$

Notice that we can represent this multiset as a matrix, in such a way that the j-th row ($1 \le j \le p$) encodes the j-th clause C_j of φ, and the columns ($1 \le i \le n$) are associated with variables. We denote by $cod_k(\varphi)$ the multiset $cod(\varphi)$ when the third index of all objects is equal to k.

The Boolean formula φ will be processed by the system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$. Next, we informally describe how that system works.

The solution proposed is inspired by the solution provided to the SAT problem in [17], consisting of the following stages:

– *Generation stage*: by applying division rules from **7.2**, all truth assignments for the variables $\{x_1, \ldots, x_n\}$ associated with φ are produced. This stage takes exactly n computation steps and at the i-th step, $1 \le i \le n$, of this stage, division rule is triggered by object $b_{i,i}$, producing two new membranes with all its remaining contents replicated in the new membranes labelled by 2. Simultaneously to these divisions, objects $t_{i,k}, f_{i,k}, T_{i,k}, F_{i,k}$ (by applying rules from **7.3**) and objects $x_{i,j,k}, \overline{x}_{i,j,k}, x{*}_{i,j,k}$ (by applying rules from **7.5**) evolve during this stage in such manner that at configuration $\mathcal{C}_n$:
  (a) There is a membrane labelled by 1 which contains n copies of object α_n.
  (b) There are 2^n membranes labelled by 2 such that each of them contains: a copy of object β, the set $cod_n(\varphi)$, the multiset $\{T_i^p, F_i^p \mid 1 \le i \le n\}$; and a different subset $\{r_{1,n}, \ldots, r_{n,n}\}$, being $r \in \{t, f\}$.
– *Preparation of enough copies for each truth assignment*: in this stage p copies (p is the number of clauses of φ) of each truth assignment are prepared, in order to allow the checking of the literal associated with each variable in each clause. This is done by means of a filtering process applied over the objects T_i and F_i (remember that we have p copies of both of them available on each of the membranes after having applied rules from **7.3** over the initial multiset). By using minimal cooperation and minimal production (applying

rules from **7.4**), objects $t_{i,k}$ (respectively, object $f_{i,k}$) are used to remove all copies of F_i (respectively, T_i). This stage takes exactly p steps, and at configuration $\mathcal{C}_{n+p}$:

(a) The root membrane (labelled by 1) contains n copies of object α_{n+p}.

(b) There are 2^n membranes labelled by 2 such that each of them contains: a copy of object β, n copies of the garbage object $\natural$, the set $cod_{n+p}(\varphi)$, and a different multiset $\{R_1^p, \dots, R_n^p\}$, being $R \in \{T, F\}$, which corresponds to the truth assignment associated to this membrane.

– *First Checking stage*: by applying rules from **7.6**, we check whether or not each clause of the input formula φ is satisfied by the truth assignments prepared in the previous stage, encoded by each membrane labelled by 2. This stage takes exactly one computation step and at configuration $\mathcal{C}_{n+p+1}$:
  (a) The root membrane (labelled by 1) contains n copies of object α_{n+p+1}.
  (b) There are 2^n membranes labelled by 2 such that each of them contains: a copy of object β, many copies of the garbage object $\natural$ (which they will not evolve in the rest of the computation), and copies of objects c_j whose presence means that clause C_j is true for the truth assignment encoded by that membrane.
– *Second Checking stage*: by applying rules from **7.7**, we check whether or not all clauses of the input formula φ are satisfied by some truth assignment encoded by a membrane labelled by 2. This stage takes exactly $p - 1$ steps and at configuration $\mathcal{C}_{n+2p}$:
  (a) The root membrane (labelled by 1) contains n copies of object α_{n+2p}.
  (b) There are 2^n membranes labelled by 2 such that each of them contains: a copy of object β, many copies of the garbage object $\natural$ (which they will not evolve in the rest of the computation), and copies of objects d_j and c_j, in such manner that the truth assignment encoded by such membrane makes true φ if and only if contains some object d_p.
– *Output stage. Negative answer:* if the input formula is not satisfiable, then any rule from **7.8** is not applicable and from $\mathcal{C}_{n+2p}$ on, no rules are applied in the system except those from **7.1** until reaching a halting configuration at $\mathcal{C}_{2n+2p+1}$. Therefore, in this case, the system answers 0.
– *Output stage. Affirmative answer:* if the input formula is satisfiable, by applying rules from **7.8** some objects γ_0 are produced at membrane labelled by 1, Due to the semantics of these membrane systems, this stage takes exactly two steps. Thus, at configuration $\mathcal{C}_{n+2p+2}$ the multiplicity of γ_0 in the skin membrane equals to the number of truth assignment of variables $\{x_1, \dots, x_n\}$ that makes true φ. Next, by applying rules from **7.9**, some objects γ_i, $0 \leq i \leq n-1$, with multiplicity 1 will be generated after, at most, $n - 1$ computation steps. Then, at configuration $\mathcal{C}_{(n+2p+2)+n-1} = \mathcal{C}_{2n+2p+1}$ at the skin membrane we have n copies of object $\alpha_{2n+2p+1}$ and some objects γ_i, $0 \leq i \leq n-1$, with multiplicity 1. By applying the second rule from **7.9**, some objects $a_{i,0}$ with multiplicity 1 are produced at that membrane. In order to make deterministic the system, objects $a_{i,0}$ evolves until $a_{i,i}$ by applying the third rules from **7.9**. Finally, the system sends to the environment the right answer according to the results of the previous stage, by applying

rules from **7.10**, for instance, object $a_{i,i}$ is released to the environment as object a_i. This stage takes, at most, n computation steps. Specifically, if object $a_{i,0}$ appears in membrane 1 at configuration $\mathcal{C}_{2n+2p+2}$ then object a_i is sent out to the environment at $i+1$-th step of this stage.

5 Main results

Theorem 1. $\#\texttt{SAT} \in \mathbf{PCMS}_{\mathcal{DAM}^0_c(mcmp,+c,-d,-n)}$.

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

(a) Every system of the family $\mathbf{\Pi}$ belongs to $\mathcal{DAM}^0_c(mcmp,+c,-d,-n)$.
(b) The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines because for each $n, p \in \mathbb{N}$, the amount of resources needed to build $\Pi(\langle n,p\rangle)$ is of a polynomial order in n and p:
- Size of the alphabet: is of the order $O(n^2 \cdot p^2)$.
- Initial number of membranes: $2 \in \Theta(1)$.
- Initial number of objects in membranes: $2np + 2n + 1 \in \Theta(n \cdot p)$.
- Number of rules: is of the order $O(n^2 \cdot p^2)$.
- Maximal number of objects involved in any rule: $3 \in \Theta(1)$.

(c) The pair (cod, s) of polynomial-time computable functions defined fulfill the following: for each input formula φ of the $\#\texttt{SAT}$ problem, $s(\varphi)$ is a natural number, $cod(\varphi)$ is an input multiset of the system $\Pi(s(\varphi))$, and for each $n \in \mathbb{N}$, $s^{-1}(n)$ is a finite set.
(d) The family $\mathbf{\Pi}$ is polynomially bounded: indeed, for each input formula φ of the $\#\texttt{SAT}$ problem, the P system $\Pi(s(\varphi)) + cod(\varphi)$ takes at most $n + 2p$ computation steps in the case of the input formula is not satisfiable and, on the contrary, takes at most $3n+2p+2$ steps, n being the number of variables of φ and p the number of clauses.
(e) The family $\mathbf{\Pi}$ is sound and complete with regard to (X, cod, s): indeed, it is informally deduced from the overview of the computations previously described.

Therefore, the family $\mathbf{\Pi}$ of P systems previously constructed solves the $\#\texttt{SAT}$ problem in polynomial time in a uniform way.

Corollary 1. $\#\mathbf{P} \subseteq \mathbf{PCMS}_{\mathcal{DAM}^0_c(mcmp,+c,-d,-n)}$.

Proof. It suffices to note that the $\#\texttt{SAT}$ problem is a $\#\mathbf{P}$-complete problem, $\#\texttt{SAT} \in \mathbf{PCMS}_{\mathcal{DAM}^0_c(mcmp,+c,-d,-n)}$, and class $\mathbf{PCMS}_{\mathcal{DAM}^0_c(mcmp,+c,-d,-n)}$ is closed under polynomial-time reduction and under complement.

6 Conclusions

In order to provide a natural framework to solve counting problems in the context of Membrane Computing, a new class of membrane systems, called *counting* *membrane systems*, is presented in this paper. The new kind of models is inspired from counting Turing machines [19] and from recognizer membrane systems [12].

The computational efficiency of the new variant has been explored. Specifically, a polynomial-time and uniform solution to the #`SAT` problem, a well-known #**P**-complete problem, is provided by using a family of counting polarizationless P systems with active membranes, without dissolution rules and division rules for non-elementary membranes but where very restrictive cooperation (minimal cooperation and minimal production) in object evolution rules is allowed.

As future works we suggest to analyze the computational efficiency of counting membrane systems from the previous class but where minimal cooperation and minimal production only is considered for communication rules (maybe only send-in rules or only send-out rules) instead of object evolution rules, following the work initiated in [18]. Besides, it would be interesting, from a computational complexity point of view, to explore the ability to use separation rules (*distribution of objects*) instead of division rules (*replication of objects*) in counting membrane systems, from a computational complexity point of view.

References

1. A. Alhazov, L. Pan. Polarizationless P systems with active membranes. *Grammars*, **7** (2004), 141-159.
2. A. Alhazov, L. Pan, Gh. Păun. Trading polarizations for labels in P systems with active membranes. *Acta Informatica*, **41**, 2-3 (2004), 111-144.
3. A. Alhazov, L. Burtseva, S. Cojocaru, Y. Rogozhin. Solving **PP**-Complete and #**P**-Complete Problems by P Systems with Active Membranes. In D. Wolfe Corne, P. Frisco, Gh. Păun, G. Rozenberg, A. Salomaa (eds.). *Membrane Computing 9th International Workshop, WMC 2008, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers. Lecture Notes in Computer Science*, **5391** (2009), 108-117.
4. M.R. Garey, D.S. Johnson. *Computers and Intractability A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.
5. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, F.J. Romero-Campero. On the power of dissolution in P systems with active membranes. In R. Freund, Gh. Păun, Gr. Rozenberg, A. Salomaa (eds.). *Membrane Computing, 6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised Selected and Invited Papers, Lecture Notes in Computer Science*, **3850** (2006), 224-240.
6. A. Leporati, G. Mauri, C. Zandron, Gh. Păun, M.J. Pérez-Jiménez. Uniform solutions to SAT and Subset Sum by spiking neural P systems. *Natural Computing*, **8**, 4 (2009), 681-702.
7. Gh. Păun. Computing with membranes, *Journal of Computer and Systems Science*, **61**, 1 (2000), 108-143.
8. Gh. Păun. Attacking **NP**-complete problems. In *Unconventional Models of Computation, UMC'2K* (I. Antoniou, C. Calude, M. J. Dinneen, eds.), Springer-Verlag, 2000, 94-115.
9. Gh. Păun. P systems with active membranes: attacking **NP**-complete problems, *Journal of Automata, Languages and Combinatorics*, **6** (2001), 75-90.

10. Gh. Păun, M.J. Pérez-Jiménez, Gr. Rozenberg. Spike trains in spiking neural P systems. *International Journal of Foundations of Computer Science*, **17**, 4 (2006), 975-1002.
11. Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez. Tissue P systems with cell division. *International Journal of Computers, Communications & Control*, Vol. **III**, 3 (2008), 295-303.
12. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, **2**, 3 (2003), 265-285.
13. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini. A polynomial complexity class in P systems using membrane division, *Journal of Automata, Languages and Combinatorics*, **11**, 4 (2006) 423-434.
14. P. Sosík, A. Rodríguez-Patón. Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences*, **73** (2007), 137–152.
15. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez. Polarizationless P systems with active membranes: Computational complexity aspects. *Journal of Automata, Languages and Combinatorics*, **21**, 1-2 (2016), 107–123.
16. L. Valencia-Cabrera, D. Orellana-Martín, A. Riscos-Núñez, M.J. Pérez-Jiménez. Minimal cooperation in polarizationless P systems with active membranes. In C. Graciani, Gh. Păun, D. Orellana-Martín, A. Riscos-Núñez, L. Valencia-Cabrera (eds.) *Proceedings of the Fourteenth Brainstorming Week on Membrane Computing*, 1-5 February, 2016, Sevilla, Spain, Fénix Editora, pp. 327-356.
17. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez. Reaching efficiency through collaboration in membrane systems: dissolution, polarization and cooperation. *Theoretical Computer Science*, in press, 2017.
18. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez. Cooperation in transport of chemical substances: A complexity approach. *Fundamenta Informaticae*, in press, 2017.
19. L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, **8**, 2 (1979), 189-201.
20. G. Zhang, M.J. Pérez-Jiménez, M. Gheorghe. *Real-Life Modelling with Membrane Computing*. Series: Emergence, Complexity and Computation, Volume 25. Springer International Publishing, 2017, X + 367 pages.

Short Papers

Communication Complexity of Distributed Tissue-like P Systems for Solving SAT Problem

Kelvin Buño[1], Henry Adorna[1,2], Linqiang Pan[2,3,⋆], and Bosheng Song[2]

[1] Department of Computer Science (Algortihms & Complexity)
University of the Philippines Diliman
Diliman 1101 Quezon City, Philippines
E-mail: kcbuno@upd.edu.ph, hnadorna@up.edu.ph
[2] Key Laboratory of Image Information Processing and Intelligent Control of Education Ministry of China
School of Automation
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
E-mail: lqpan@mail.hust.edu.cn, boshengsong@hust.edu.cn
[3] School of Electric and Information Engineering
Zhengzhou University of Light Industry
Zhengzhou 450002, China

Abstract. In this work, we consider dP systems, where the input set of the problem is partitioned and each part enters the system through distinct components. We present a 2-component solution to the SAT problem using dP systems, where the components are tissue P systems with evolutional symport/antiport rules and cell division, and the input multiset is partitioned based on the clauses of the boolean formula. We show that this 2-component solution requires only one computation step for intercomponent communication, and that the number of rules and the number of objects communicated depend on the number of satisfying assignment to the boolean formula. Additionally, we show that the 2-component solution can reduce the length of the computation by half if the number of clauses is square the number of variables.

Keywords: Membrane computing, Dynamical communication measure, dP scheme, dP system, Tissue P system, Satisfiability problem

1 Introduction

Membrane computing is an innovative computational paradigm inspired from the architecture and the functioning of living cells, as well as from the organization of cells in tissues, organs (brain included) or other populations of cells. This area was initiated in 1998 by Gh. Păun [1], and its literature has grown very fast both on theoretical investigation [2–4] and applications [5, 6]. The computing models

⋆ Corresponding author.

investigated in membrane computing are called *P systems*. A comprehensive information of membrane computing can be found in [7].

A tissue-like P system consists of cells that is described by a directed graph, where cells are nodes of a graph, an edge between two nodes corresponds to a communication channel between cells. If a communication channel between two cells or between a cell and the environment exists, then they can communicate [8]. Object located in cells can exchange between these cells by using communication (symport/antiport) rules. Symport rules move objects of multiset through a membrane in one direction; whereas antiport rules move objects of multisets across a membrane in opposite directions.

In standard tissue P systems, objects are never modified in the process of communication, they just change their place within the system. In [9], tissue-like P systems with evolutional symport/antiport rules and cell division were proposed, where objects can be evolved when moving from one region to another region. It was shown in [9] that the SAT problem can be solved by tissue P systems with evolutional symport/antiport rules and cell division, and this solution was evaluated and analysed based on the amount of time.

Communication complexity of P systems was suggested as a research topic in [10] and [11]. A related measure for symport/antiport P systems namely *communication difference (Comdif)* was discussed in [12]. *Comdif* is the difference of numbers of input and output objects in an antiport rule. This measure of communication is static. The notion of dynamical parameters to measure the communication complexity of P systems was initiated in [13], where the number of communication steps and the number of communication rules used during a computation are counted.

Distributed P systems (dP systems, for short) were introduced in [14] in the light of possibly applying directly the idea of communication complexity started in 1979 by Yao in [15] to membrane computing. In [14], the authors adopted the communication complexity measures introduced in [13]. In [16], SN dP systems were proposed, and languages generated by SN dP systems was investigated. Controlled rewriting distributed P systems were proposed in [17], it was shown that controlled rewriting dP systems are Turing universal.

In this work, we consider a variant of tissue P systems with evolutional symport/antiport rules, called *distributed tissue P systems with evolutional symport/antiport rules*. We present a 2-component solution to the SAT problem using distributed tissue P systems with evolutional symport/antiport rules and cell division, where the input set of the problem is partitioned and each part enters the system through distinct components. We show that the 2-component solution requires only one computation step for intercomponent communication, and that the number of rules and the number of objects communicated depend on the number of satisfying assignment to the boolean formula. Additionally, we show that the 2-component solution can reduce the length of the computation by half if the number of variables and number of clauses of the boolean formula follow a given ratio.

2 dP Scheme and Tissue-like P Systems

The reader is assumed to be familiar with the fundamentals of formal language theory. Let Σ be an alphabet, the Kleene closure of Σ, denoted as Σ^*, is the set of all finite words over the alphabet Σ. We denote by λ the word of length zero (the empty word), and by Σ^+ the set of all non-empty words. A multiset M over Σ, is a mapping from Σ to the set of non-negative integers.

2.1 dP Scheme

We recall dP scheme, which was introduced in [14].

Definition 1 [14] *A* **k-dP scheme,** *$k \geq 2$ is a tuple*

$$k\text{-}\Delta_\Pi = (\Gamma, \Pi_1, \Pi_2, \dots \Pi_k, R),$$

where:

1. *Π is a fixed variant of P systems.*
2. *Γ is an alphabet of objects in the whole system Δ.*
3. *Π_i for $i = 1, 2, \dots, k$ are some copies fixed variant of P systems with set of objects in Γ. Skin membranes or local environments of each P system will be labelled injectively as $s_1, s_2, \dots, s_k$.*
4. *R is set of finite rules of the form*

$$(s_i, u/v, s_j),$$

where $1 \leq i, j \leq k$, for $i \neq j$, and $u, v \in \Gamma^*$, such that $uv \neq \lambda$. We denote by $|uv|$ the weight of the rule $(s_i, u/v, s_j)$. This antiport-like communication rule is called **inter-component communication rule.**

The details on how this distributed model works could be found in [14].

As the input set is partitioned into multiple components, another consideration of the computation in dP schemes is the communication cost. Communication measures are introduced in [14] for analysing the communication cost of the system. These are defined as follows:

Let Δ be a dP scheme, $\delta : \delta_0 \Rightarrow \delta_1 \Rightarrow \cdots \Rightarrow \delta_h$ be a halting computation in Δ, and R the set of intercomponent communication rules, with each rule $r \in R$ of the form $(s_i, u/v, s_j)$. Then for each $i = 0, 1, \dots, h-1$, we have the following parameters:

- $ComN(\delta_i \Rightarrow \delta_{i+1}) = 1$, if at least one intercomponent communication rule, $r \in R$, is used in this transition, and 0 otherwise;
- $ComR(\delta_i \Rightarrow \delta_{i+1}) =$ the number of intercomponent communication rules used in this transition;
- $ComW(\delta_i \Rightarrow \delta_{i+1}) =$ the sum of weights of all intercomponent communication rules used in this transition. Recall that the weight of a rule $r \in R$ is $|uv| >= 1$.

These parameters can also be used to measure computations, results of computations, systems, and languages. We denote the set of strings accepted by Δ as $L(\Delta)$. For $ComX \in \{ComN, ComR, ComW\}$, we define:

- $ComX(\delta) = \sum_{i=0}^{h-1} ComX(\delta_i \Rightarrow \delta_{i+1})$, for δ which is a halting computation.
- $ComX(w, \Delta) = min\{ComX(\delta) \mid \delta$ is a computation of Δ that accepts the string $w\}$.
- $ComX(\Delta) = max\{ComX(w, \Delta) \mid w \in L(\Delta)\}$.

With respect to these communication measures, the idea of parallelizability for dP systems is also introduced in [14], but while the definition is for dP systems with P automata (see [18]) components, we can also use it for dP systems using other classes of P systems as components.

A language $L \subseteq V^*$ is said to be (n, m)-weakly $ComX$ parallelizable, for some $n \geq 2, m \geq 1$ and $X \in \{N, R, W\}$, if there is a dP system Δ with n components and there is a finite subset F_Δ of L such that each string $x \in L - F_\Delta$ can be written as $x = x_1 x_2 \ldots x_n$, each component Π_i of Δ takes as input the string $x_i, i \leq i \leq n$, and the string x is accepted by Δ by a halting computation δ such that $ComX(\delta) \leq m$. A language L is said to be *weakly ComX parallelizable* if it is (n, m)-weakly Com$X$ parallelizable for some $n \geq 2, m \geq 1$.

2.2 Distributed Tissue-like P Systems with Evolutional Symport/Antiport Rules and Cell Division

Distributed P scheme can be viewed as a template for constructing distributed P systems once a class of P systems is chosen as its components. In this study, we look into using the tissue-like P systems with evolutional symport/antiport rules and cell division rules (TPec) as components of our dP system. TPec is formally defined as follows:

Definition 2 [9] *A* **tissue P system (of degree $q \geq 1$) with evolutional symport/antiport rule and cell division** *is a tuple*

$$\Pi = (\Gamma, \mathcal{E}, M_1, M_2, \ldots, M_q, R, i_{out}),$$

where:

1. *Γ is an alphabet of objects.*
2. *$\mathcal{E} \subseteq \Gamma$, is a set of objects initially located at the environment.*
3. *M_i, $1 \leq i \leq q$ are finite multisets over Γ.*
4. *R is a finite set of rules of the following forms:*
   (a) *Evolutional communication rules:*
      i. *Evolutional symport rules: $[u]_i[\,]_j \rightarrow [\,]_i[u']_j$, for $1 < i \leq q, 0 < j \leq q, i \neq j; u \in \Gamma^+, u' \in \Gamma^*$ or $i = o, 1 < j \leq q; u \in \Gamma^+, u' \in \Gamma^*$, and there exists at least one object $a \in \Gamma \setminus E$, which is in cell $i = 0$. The length of an evolutional symport rule is defined as $|u| + |u'|$.*

   ii. Evolutional antiport rule: $[u]_i[v]_j \rightarrow [v']_i[u']_j$, *where* $0 \leq i \neq j \leq q$, $u, v \in \Gamma^+, u', v' \in \Gamma^*$. *The length of an evolutional antiport rule is defined as* $|u| + |u'| + |v| + |v'|$.

  (b) Division rules:

   i. $[a]_i \rightarrow [b]_i[c]_i$, *where* $i \in \{1, 2, \ldots, q\}$, $i \neq i_{out}, a, b, c \in \Gamma$.

5. $i_{out} \in \{1, 2, , \ldots, q\}$.

Details on the mechanism on how this variant performs its computation could be found in [9].

We now introduce our dP system model. This dP system uses tissue P systems with evolutional symport/antiport rules and cell division rules, as our fixed P systems for our dP scheme.

Definition 3 *A* **k-distributed tissue P system with evolutional symport/antiport rules and cell division rules** *or* **k-dTPec system** *as follows:*

$$k\text{-}\Delta_{TPec} = (\Gamma, \Pi_1, \Pi_2, \ldots \Pi_k, R, i_{out}),$$

where:

1. Γ *is an alphabet of objects in the whole system* Δ.
2. Π_i $(i = 1, 2, \ldots, k)$ *are tissue P systems with evolutional symport/antiport rules and cell division. Each* Π_i *has* Γ *as the alphabet of objects. Each cell of* Π_i *will be labeled* $\langle i, j \rangle$, *where* $j = 1, 2, \ldots, d_i$, *and* d_i *denotes the number of cells of* Π_i. *The external region, or the environment, are different for each component. We will refer to the environment of component* i *as the local environment of* i *and is denoted by the label* $\langle i, 0 \rangle$.
3. R *is a set of finite intercomponent communication rules of the form:*

$$(\langle i, 0 \rangle, u/v, \langle j, 0 \rangle),$$

   where:
   - $\langle i, 0 \rangle$ *and* $\langle j, 0 \rangle$ *are the local environments of components* i *and* j, *respectively,*
   - $u, v \in \Gamma^*$, *and* $|uv| \geq 1$.
4. i_{out} *is the component of the dTPec designated as the output component. Only the objects produced in the output region/cell of* $\Pi_{i_{out}}$ *are considered as output in a halting computation of the dTPec.*

For dTPec, we restrict intercomponent communication to their local environment. No evolution of objects occur during intercomponent communication. The restriction of the intercomponent communication to the local environment of each component is made to follow the original defintion of intercomponent communication rules in [14] where intercomponent communication only occurs between the skin membranes of each component.

3 dTPec and SAT

The satisfiability problem (SAT) is a decision problem where given a Boolean formula in conjunctive normal form, does there exist an assignment to its variables such that the formula evaluates to true.

In [9], a recognizer TPec was used to provide a polynomial time solution to SAT. A recognizer TPec is formally defined as:

Definition 4 [9] *A* **recognizer tissue P system with evolutional symport/antiport rules and cell division of degree** $q \geq 1$ *is a tuple*

$$\Pi = (\Gamma, \Sigma, \mathcal{E}, \mathcal{M}_1, ..., \mathcal{M}_q, R, i_{in}, i_{out}),$$

where:

- Γ *has distinguished objects* **yes** *and* **no***,*
- $\Sigma \subset \Gamma$ *is the input alphabet,*
- $\mathcal{M}_1, ..., \mathcal{M}_q$ *are finite multisets over* $\Gamma \setminus \Sigma$*,*
- $i_{in} \in \{1, ..., q\}$ *is the label of the input cell, and* $i_{out} = 0$*,*
- *all computations halt,*
- *either a* **yes** *or a* **no** *is released into the environment at the last step of any computation.*

We restate the result from [9] as the following theorem:

Theorem 1. *[9] Let* $PMC_{TDEC(k)}$ *be the set of all decision problems solvable in a uniform way and polynomial time by means of recognizer tissue P systems with cell division and evolutional communication rules of length at most* k*. Then,* $SAT \in PMC_{TDEC(k)}$.

3.1 Solving SAT with dTPec

The construction of the dTPec solving SAT will be based on the solution of [9]. We present the main result of this study.

Theorem 2. *The satisfiability problem (SAT) with boolean formula* φ *of* n *variables, and* m *clauses is* $(2, r)$*-weakly ComX parallelizable, where* $(r, ComX) \in \{(1, ComN), (S, ComR), (2S, ComW)\}$*, and* S *is the number of satisfying assignments to* φ*.*

Proof:
Consider a boolean formula of $m \geq 1$ clauses, and $n \geq 1$ variables, $\varphi = C_1 \wedge \ldots \wedge C_m$, with $C_i = y_{i,1} \vee \ldots \vee y_{i,p_i}$, for some $p_i \geq 1$, and $y_{i,j} \in \{x_k, \overline{x}_k \mid 1 \leq k \leq n\}$, for each $1 \leq i \leq m$, and $1 \leq j \leq p_i$. The boolean formula φ is encoded as the multiset $\{x_{i,j} \mid x_j \in C_i\} \cup \{\overline{x}_{i,j} \mid \overline{x}_j \in C_i\}$. The multiset is then used as input for the recognizer TPec.

We partition the encoded multiset of φ, $\Sigma = \{x_{i,j} \mid x_j \in C_i\} \cup \{\overline{x}_{i,j} \mid \overline{x}_j \in C_i\}$ into two. We partition the elements of the multiset according to their clause indices and goes as follows:

– $P_1 = \{x_{i,j}, \overline{x}_{i,j} \mid 1 \leq i \leq m_1\}$, and
– $P_2 = \{x_{i,j}, \overline{x}_{i,j} \mid m_1 + 1 \leq i \leq m\}$,

where $m_1 = \lceil \frac{m}{2} \rceil$ denotes the number of clauses assigned to Π_1. Likewise, we denote by $m_2 = m - m_1$ the number of clauses assigned to Π_2. Note here that $m_1 \geq m_2$, but the difference is at most 1. Also, note that elements with the same clause index are placed together in either P_1 or P_2, but never in both at the same time. Using this partition, we can then construct a 2-component dTPec to solve the SAT problem.

$$\Delta_{TPec} = (\Gamma, \Pi_1, \Pi_2, i_{out}, R_\Delta),$$

where:

– $\Gamma = \Sigma \cup \{a_i, t_i, f_i \mid 1 \leq i \leq m\} \cup \{e_{i,j}, e'_{i,j}, \overline{e}_{i,j}, \overline{e'}_{i,j}, E_{i,j}, \overline{E}_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq m\} \cup \{d_{i,j,k}, d'_{i,j,k}, \overline{d}_{i,j,k}, \overline{d'}_{i,j,k} \mid 1 \leq i \leq n, 1 \leq j \leq m, 1 \leq k \leq n-1\} \cup \{b_j, c_j, \overline{c}_j, c'_j, E_j \mid 1 \leq j \leq m\} \cup \{b_{j,k}, b'_{j,k} \mid 1 \leq j \leq m, 1 \leq k \leq n-1\} \cup \{\alpha_i, \alpha'_i \mid 0 \leq i \leq 2n+3m\} \cup \{d, E_{m+1}, \alpha_{2n+3m+1}, \mathbf{yes}, \mathbf{no}\} \cup \{p_{0,i}, p'_{0,i} \mid 0 \leq i \leq n\} \cup \{p_i \mid 0 \leq i \leq n+1\} \cup \{q_w \mid w \in \bigcup_{0 \leq k \leq n} \{0,1\}^k\} \cup \{q_{w,1}, q_{w,2} \mid w \in \{0,1\}^n\}$,
– $\Sigma = \{x_{i,j} \mid x_j \in C_i\} \cup \{\overline{x}_{i,j} \mid \overline{x}_j \in C_i\}$,
– Each recognizer tissue P system, $\Pi_\beta(\langle n, m \rangle)$, $\beta = 1, 2$, is defined as:

$$\Pi_\beta(\langle n, m \rangle) = (\Gamma, \Sigma, \mathcal{E}, \mathcal{M}_{\beta,1}, \mathcal{M}_{\beta,2}, \mathcal{M}_{\beta,3}, R_\beta, i_{in} = 2, i_{out} = 0),$$

where:
- $\mathcal{E} = \{\alpha'_i \mid 0 \leq i \leq 3n + 3_{m_1} + 3\}$,
- $\mathcal{M}_1 = \{a_1, \dots, a_n, E_1\}$, $\mathcal{M}_2 = \{b_1, \dots, b_n\}$, and $\mathcal{M}_3 = \{d, \alpha_0, p_{0,0}\}$,
- R_β is the set of rules:
(Rules $r_{1,i}$ to $r_{25,i,j}$ are mostly unchanged from [9], except that now, the index j has bounds $1 \leq j \leq m_1$ for Π_1, and $1 \leq j \leq m_2$, for Π_2).

$r_{26,i} \equiv [\ \alpha_i\ ]_{\langle \beta,3 \rangle} [\ \alpha'_i\ ]_{\langle \beta,0 \rangle} \rightarrow [\ \alpha_{i+1}]_{\langle \beta,3 \rangle} [\ ]_{\langle \beta,0 \rangle}$, $0 \leq i \leq 3n + 3m_1 + 4$
(Rule r_{27} is removed. Rule r_{28} is replaced by rule r_{34}.)

$r_{29} \equiv [\ \alpha_{3n+3m_1+5} d\ ]_{\langle 1,3 \rangle} [\ ]_{\langle 1,0 \rangle} \rightarrow [\ ]_{\langle 1,3 \rangle} [\ \mathbf{no}\ ]_{\langle 1,0 \rangle}$
(Rule r_{29} is used at the end of a halting computation to output a negative decision, indicating no satisfying assignment.)

$r_{30} \equiv [\ E_{m_\beta+1}\ ]_{\langle \beta,1 \rangle} [\ p_0\ ]_{\langle \beta,3 \rangle} \rightarrow [\ q_\lambda\ ]_{\langle \beta,1 \rangle} [\ p_1\ ]_{\langle \beta,3 \rangle}$
$r_{31,i} \equiv [\ t_i q_w\ ]_{\langle \beta,1 \rangle} [\ p_i\ ]_{\langle \beta,3 \rangle} \rightarrow [\ q_{w1}\ ]_{\langle \beta,1 \rangle} [\ p_{i+1}\ ]_{\langle \beta,3 \rangle}$, $1 \leq i \leq n$, $w \in \bigcup_{0 \leq k \leq n-1} \{0,1\}^k$
$r_{32,i} \equiv [\ f_i q_w\ ]_{\langle \beta,1 \rangle} [\ p_i\ ]_{\langle \beta,3 \rangle} \rightarrow [\ q_{w0}\ ]_{\langle \beta,1 \rangle} [\ p_{i+1}\ ]_{\langle \beta,3 \rangle}$, $1 \leq i \leq n$, $w \in \bigcup_{0 \leq k \leq n-1} \{0,1\}^k$
(Rules $r_{30}, r_{31,i}, r_{32,i}$ encode the generated truth assignment that satisfies the clauses assigned to the component. We refer to the produced

object as the communication object containing a 'candidate' satisfying assignment.)

$r_{33} \equiv [\ q_w\ ]_{\langle\beta,1\rangle}[\ p_{n+1}\ ]_{\langle\beta,3\rangle} \rightarrow [\ ]_{\langle\beta,1\rangle}[\ q_{w,\beta}\ ]_{\langle\beta,3\rangle},\ w \in \{0,1\}^n$
$r_{34} \equiv [\ ]_{\langle\beta,0\rangle}[\ q_{w,\beta}\ ]_{\langle\beta,3\rangle} \rightarrow [\ q_{w,\beta}\ ]_{\langle\beta,0\rangle}[\ ]_{\langle\beta,3\rangle},\ w \in \{0,1\}^n$
(Rule r_{33} and r_{34} is used to transfer the object containing the encoded candidate satisfying assignment to the local environment to enable intercomponent communications.)

$r_{35} \equiv [\ q_{w,2}\ ]_{\langle1,0\rangle}[\ d\ ]_{\langle1,3\rangle} \rightarrow [\ \mathrm{q_w}\ \text{yes}\ ]_{\langle1,0\rangle}[\ ]_{\langle1,3\rangle}$
(When a communication occurs, rule r_{35} produces the positive decision and an object containing the encoded solution.)

$r_{36,i} \equiv [\ p_{0,i}\ ]_{\langle\beta,3\rangle}[\ ]_{\langle\beta,0\rangle} \rightarrow [\ ]_{\langle\beta,3\rangle}[\ p'_{0,i+1}]_{\langle\beta,0\rangle},\ 0 \le i \le n-1$
$r_{37,i} \equiv [\ ]_{\langle\beta,3\rangle}[\ p'_{0,i}\ ]_{\langle\beta,0\rangle} \rightarrow [\ p^2_{0,i}\ ]_{\langle\beta,3\rangle}[\ ]_{\langle\beta,0\rangle},\ 1 \le i \le n-1$
$r_{38} \equiv [\ ]_{\langle\beta,3\rangle}[\ p'_{0,n}\ ]_{\langle\beta,0\rangle} \rightarrow [\ p^2_0\ ]_{\langle\beta,3\rangle}[\ ]_{\langle\beta,0\rangle}$

- $i_{out} = \Pi_1$. We designate component Π_1 as the output component of Δ_{TPEC}. Only the objects produced by Π_1 in its environment at the end of a halting computation will be considered as the output of system Δ_{TPEC}.
- R_Δ is the set of inter-component communication rules:
  - $\{(\langle1,0\rangle, q_{w,1}/q_{w,2}, \langle2,0\rangle) \mid w \in \{0.1\}^n\}$.

The computation of Δ_{TPEC} is separated into five stages: (1) Generation, (2) Checking, (3) Encoding, (4) Communication, and (5) Output. The two $TPec$ components of Δ_{TPec} are based on the solution of [9], using rules r_1 upto r_{25} for the Generation and Checking Stage.

Lemma 1. *Given the partitioned encoded multiset of φ, the TPec components Π_1 and Π_2 of Δ_{TPec} produces a satisfying assignment for each partition of φ, if there are any, in $2n + 3m_1$ steps.*

This lemma is a result of Theorem 1. Each component will produce a satisfying assignment for all the clauses assigned to them. As each component can only evaluate half the clauses of φ, in order for Δ_{TPec} to decide if φ is satisfiable, each component must communicate with each other their respective solution(s).

Before any communication occurs, Π_1 will encode each existing candidate assignment into single object, $q_{w,1}$, where $w \in \{0,1\}^n$ is a representation of the candidate assignment.

Lemma 2. *Let w be a binary string of length n. w encodes a satisfying assignment to φ if, and only if, $q_{w,1}$ and $q_{w,2}$ is produced in cell $\langle1,0\rangle$, and cell $\langle2,0\rangle$, respectively, after $3n + 3m_1 + 3$ steps.*

Proof. Let $w = w_1 \dots w_n$ be a binary string of length n. For the first half of the proof, suppose that w encodes a satisfying assignment to φ. Since all possible assignments are generated by step $2n$, there will exist a cell $\langle1,1\rangle$ in Π_1 (cell $\langle2,1\rangle$ in Π_2) containing the objects $a_1, ..., a_n$, where for each $1 \le i \le n$, $a_i \in \{t_i, f_i\}$, such that if $w_i = 1$, then $a_i = t_i$, and if $w_i = 0$, then $a_i = f_i$. For ease of reference, let us refer to this cell as $\langle 1, 1\rangle_w$ ($\langle 2, 1\rangle_w$ for Π_2).

Since w encodes a satisfying assignment, by Lemma 1, the cell $\langle 1, 1\rangle_w$ ($\langle 2, 1\rangle_w$) will produce object E_{m_1+1} at step $2n+3m_1$. The object q_w is produced in $\langle 1, 1\rangle_w$ ($\langle 2, 1\rangle_w$) at step $3n + 3m_1 + 1$ using rules r_{30}, r_{31}, and r_{32}. For the next two steps, using rules r_{33} and r_{34} in order, object $q_{w,1}$ is produced in $\langle 1, 0\rangle$ in Π_1, and likewise, object $q_{w,2}$ is produced in $\langle 2, 0\rangle$ in Π_2. This completes half the proof.

Now, suppose that binary string $w = w_1 \dots w_n$ does not encode a satisfying assignment. We have to show that either $q_{w,1}$ is not produced in $\langle 1, 0\rangle$, or $q_{w,2}$ is not produced in $\langle 2, 0\rangle$. Since w does not encode a satisfying assignment, then for some $\beta \in \{1, 2\}$, the cell $\langle \beta, 1\rangle_w$ will fail to produce object E_{m_1+1}. This will prevent cell $\langle \beta, 1\rangle_w$ from producing q_w, and hence, $q_{w,\beta}$ will never be introduced in $\langle \beta, 0\rangle$. Therefore, either $q_{w,1}$ is not produced in $\langle 1, 0\rangle$, or $q_{w,2}$ is not produced in $\langle 2, 0$ by the end of the encoding phase. This completes the proof for Lemma 2. □

At step $3n + 3m_1 + 4$, if there exists an object $q_{w,1}$ in the local environment $\langle 1, 0\rangle$, and an object $q_{w',2}$ in the local environment $\langle 2, 0\rangle$, and $w = w'$, then an intercomponent communication rule, $(\langle 1, 0\rangle, q_{w,1}/q_{w,2}, \langle 2, 0\rangle)$ is applied.

At step $3n + 3m_1 + 5$, if there is at least one object $q_{w,2}$ in the local environment $\langle 1, 0\rangle$, for some $w \in \{0, 1\}^n$, then using rule r_{35}, Π_1 will produce exactly two objects in its output region $\langle 1, 0\rangle$: **yes** and one q_w (Only one q_w is produced even if there are many solutions). Object **yes** is used to indicate that φ has a satisfying assignment, and object q_w indicates that the assignment encoded by w is a satisfying assignment.

Otherwise, if, for any $w \in \{0, 1\}^n$, there is no object $q_{w,2}$ in the local environment $\langle 1, 0\rangle$, then at step $3n + 3m_1 + 6$, using rule 29, an object **no** is produced in the output region to indicate that φ has no satisfying assignment.

For the analysis of the communication cost, observe that Δ_{TPec} only needs one step to perform all intercomponent communications. It is done in step $3n + 3m_1 + 4$. Communication only happens when φ has a satisfying assignment. Otherwise, no communication occurs.

Note as well that a satisfying assignment for φ may not be unique, and as such, multiple $q_{w,2}$ objects may appear in $\langle 1, 0\rangle$. All of these are communicated during step $3n + 3m_1 + 4$, and we denote the number of satisfying assignment to φ as S. The number of communication rules used is S, and the number of objects communicated is $2S$. We summarize these communication analysis as follows:

Lemma 3. *Given Δ_{TPec} solving SAT, the communication measures are as follows: $ComN(\Delta_{TPec}) = 1$, $ComR(\Delta_{TPec}) = S$, and $ComW(\Delta_{TPec}) = 2S$.*

To summarize, Lemma 1 guarantees that a satisfying assignment is produced if the input boolean formula φ is satisfiable. Lemma 2 guarantees that the object containing the encoding of the satisfying assignment will be communicated between the two components. And finally, Lemma 3 gives the communication cost of solving SAT using Δ_{TPec}. This completes the proof of Theorem 2. □

Comparing the running times of the solution presented in [9] and $\varDelta_{TPec}$, let $TIME_{\varPi(\langle n,m\rangle)}(n,m)$ denotes the running time of $\varPi_(\langle n,m\rangle)$, which is $2n+3m+2$. Let $TIME_{\varDelta_{TPEC}}(n,m)$ denotes the running time of $\varDelta_{TPEC}$, which is $3n+3m_1+5$, where $m_1 = \lceil \frac{m}{2} \rceil$. Since $m_1 = \frac{m}{2}$, $TIME_{\varDelta_{TPEC}}(n,m) = 3n + 1.5m + 5$.

The running time of $\varDelta_{TPEC}$ cuts down the number of steps of the checking phase by half, but increases the number of steps by $n+3$ steps due to the encoding phase and communication phase. For easier comparison, we fix the number of variables, n, and express the number of clauses, m, in terms of m.

To achieve a speed-up ratio of at least 2, the checking phase must dominate the running time of the solution. At $m = n^2$, the speed-up ratio becomes:

$$\lim_{n\to\infty} \frac{TIME_{\varPi(\langle n,m\rangle)}(n,m)}{TIME_{\varDelta_{TPEC}}(n,m)} = \frac{2n+3n^2+2}{3n+1.5n^2+5} = 2.$$

This shows that $\varDelta_{TPec}$ will only compute in half the required number of steps if $m \geq n^2$. Note that having $m < n$ will make $\varDelta_{TPec}$ compute slower.

4 Final Remarks

In this work, we have presented a solution to SAT using a distributed version of the tissue-like P systems with evolutional symport/antiport rules and cell division presented in [9]. We refer to this distributed version of TPec as distributed TPec (dTPec). The input multiset was partitioned into two, based on the clause indices of the elements of the multiset. The number of propositions per clause does not affect the length of the computation, but it will affect the number of elements in each partition. The partition only guarantees that the number of clauses each component of the dTPec processes is half the total number of clauses of the boolean formula.

This partition does not immediately guarantee a reduction in the length of the computation as compared to the non-distributed solution. The distributed version will only reduce the length of the computation by half if the number of clauses is at least square the number of variables of the given boolean formula. This is because the dTPec solution only reduces the number of clauses processed by half, but not the number of variables. The ratio of m and n needed for an efficient speed-up would differ if a different partition was chosen for the dTPec solution.

Acknowledgements

The work was supported by National Natural Science Foundation of China (61320106005, 61602192 and 61033003), China Postdoctoral Science Foundation (2016M600592, 2017T100554), and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012).

References

1. Păun, G.: Computing with membranes. Journal of Computer and System Sciences **61**(1) (2000) 108 – 143
2. Gheorghe, M., Ipate, F., Konur, S.: Testing based on identifiable P systems using cover automata and x-machines. Information Sciences **372** (2016) 565–578
3. Song, T., Pan, L., Păun, G.: Asynchronous spiking neural P systems with local synchronization. Information Sciences **219** (2013) 197–207
4. Sosík, P., Langer, M.: Small (purely) catalytic P systems simulating register machines. Theoretical Computer Science **623** (2016) 65–74
5. Peng, H., Wang, J., Pérez-Jiménez, M.J., Riscos-Núñez, A.: An unsupervised learning algorithm for membrane computing. Information Sciences **304** (2015) 80–91
6. Colomer, M., Margalida, A., Valencia, 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** (2014) 116–126
7. Păun, G., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
8. Păun, A., Păun, G.: The power of communication: P systems with symport/antiport. New Generation Computing **20**(3) (2002) 295–305
9. Song, B., Zhang, C., Pan, L.: Tissue-like P systems with evolutional symport/antiport rules. Information Sciences **378** (2017) 177–193
10. Csuhaj-Varjú, E.: P automata. In Mauri, G., Păun, G., Pérez-Jiménez, M.J., Rozenberg, G., Salomaa, A., eds.: Membrane Computing: 5th International Workshop, WMC 2004, Milan, Italy, June 14-16, 2004. Springer Berlin Heidelberg, Berlin, Heidelberg (2005) 19–35
11. Păun, G.: Further open problems in membrane computing, brainstorming week on membrane computing. In: Sevilla University. (2004) 354–365
12. Csuhaj-Varjú, E., Margenstern, M., Vaszil, G., Verlan, S.: On small universal antiport P systems. Theoretical Computer Science **372**(2-3) (2007) 152–164
13. Adorna, H., Păun, G., Pérez-Jiménez, M.J.: On communication complexity in evolution-communication P systems. Romanian Journal of Information Science and Technology **13**(2) (2010) 113–130
14. Păun, G., Pérez-Jiménez, M.J.: Solving problems in a distributed way in membrane computing: dP systems. International Journal of Computers, Communications and Control **5** (2010) 238–250
15. Yao, A.C.C.: Some complexity questions related to distributive computing(preliminary report). In: Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing. STOC '79, New York, NY, USA, ACM (1979) 209–213
16. Ionescu, M., Păun, G., Pérez-Jiménez, M.J., Yokomori, T.: Spiking neural dP systems. Fundamenta Informaticae **111**(4) (2011) 423–436
17. Buño, K.C., Adorna, H.N.: Controlled rewriting distributed P systems. In: Theory and Practice of Computation. Springer (2012) 46–61
18. Csuhaj-Varjú, E., Vaszil, G.: P automata or purely communicating accepting P systems. In Păun, G., Rozenberg, G., Salomaa, A., Zandron, C., eds.: Membrane Computing: LNCS vol 2597. Springer Berlin Heidelberg (2003) 219–233

A Simulation of Transition P Systems by Numerical P Systems with Migrating Variables

Nestine Hope S. Hernandez, Henry N. Adorna

Algorithms & Complexity Lab, Department of Computer Science
University of the Philippines, Diliman, Quezon City, Philippines
E-mail: nshernandez@dcs.upd.edu.ph, hnadorna@dcs.upd.edu.ph

Abstract. Transition P systems and Numerical P systems are two typical models of computation investigated in Membrane computing. In this work, we provide a relation between Transition P systems with noncooperative rules (or nTP systems) and Numerical P systems with migrating variables (or MNP systems). Specifically, we show that there exists a simulation of nTP systems with exhaustive use of rules having concentric membrane structure by MNP systems operating in oneP mode.

Keywords: Membrane Computing, noncooperative Transition P systems, Numerical P systems with migrating variables, simulation

1 Introduction

Membrane computing is a bio-inspired branch of natural computation that abstracts computing models from the structure and functioning of living cells and from the organization of cells in tissues or higher order structures. These computational models are known as P systems. Their most basic features are : (1) the membrane structure which can be cell-like, tissue-like or neural-like (2) regions containing multisets of objects and (3) rules inspired from the biochemistry of the cell which dictates how the objects are processed. These computing models are distributed, as evident in the compartmentalization defined by membranes; and parallel, as rules are applied simultaneously in all regions to all objects that can evolve with respect to some application mode. More details can be found in [11,12,10].

For the past two decades, several membrane computing models have been introduced and extensively studied. One of these earlier models are Transition P systems (TP systems, in short) [9]. In contrast, Numerical P systems (NP systems, in short) were introduced only recently [15] and are not yet explored as much. Though the two models have similar membrane structure (cell-like), they differ in the type of objects they contain, the type of output they produce and also in the modes of application of their rules. These differences will be evident when these two models will be discussed in the next section.

In light of these differences, it is desirable to find a relation between these two computing models (and even with other membrane systems). Relations between different P systems and/or computing models are abundant in literature [3,1,7]. In this work, we provide a simulation relation over their configurations similar to [8,3]. A specific motivation for this is embodied in the following: suppose we have a P system Π_{D} composed of modules of various families of P systems. If however a given module T becomes non-operational, how can we replace T with a module N such that N interacts with Π_{D} or its components similar to the way T does?

This paper is organized as follows: Section 2 presents the definitions of nTP and MNP systems, the mode of operation we use in this work, and the notion of simulation. Section 3 provides our main result, which is the simulation of nTP systems by MNP systems, using the application mode discussed in section 2. We provide directions for further work as well as the final remarks in Section 4.

2 Preliminaries

Readers are assumed to be familiar with the basics of membrane computing and formal language theory. For a good introductory reading, please refer to [2] and [5], respectively. Recent results and information on membrane computing can also be found in the P systems webpage at `http://ppage.psystems.eu/`. We briefly mention here some notions which will be useful throughout the paper.

We denote the set of all natural numbers as $\mathbb{N} = \{1, 2, 3 \ldots\}$. Let V be an alphabet, V^* is the free monoid over V with respect to concatenation and the identity element λ (the empty string). The length of a string $w \in V^*$ is denoted by $|w|$. The number of symbols a present in a string w is denoted by $|w|_a$. If a is a symbol in V, $a^0 = \lambda$.

We give the definition of a noncooperative transition P (nTP) system without dissolution as defined in [4]:

Definition 1. *A noncooperative transition P (nTP) system without dissolution of degree $m \geq 1$ is a tuple of the form*

$$\Pi_{nTP} = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_{out})$$

1. m is the total number of membranes;
2. O is the alphabet of objects;
3. μ is a hierarchical (cell-like) membrane structure with m membranes which can be denoted by a set of m paired square brackets with labels in $\{1, 2, \ldots, m\}$;
4. $w_1, \ldots, w_m$ are strings over O^* where w_i, $1 \leq i \leq m$, denotes the initial multiset of objects present in region i (region bounded by membrane i).;
5. $R_1, \ldots, R_m$ are sets of evolution rules, each associated with a region i in μ;
   – An evolution rule $r \in R_i$, $1 \leq i \leq m$, is of the form $a \to v$, $a \in O$ and $v \in (O \times Tar_i)^*$ where $Tar_i = \{here, out\} \cup \{in_j | j \text{ is a membrane contained in } i\}$. If a copy of object a is present in region i, we say that rule r can be applied to evolve it.
   Upon application of rule r, the object a is consumed (removed from region i) and a multiset of objects given in v are produced at the next

time step. The symbols *here*, *out* and in_j are target commands indicating the destination of the objects produced, that is, they remain in region i or sent to the region containing i or placed in region j (a region contained in i). As a short-hand notation, we use strings in place of objects that have the same target indication. Thus, if v contains (v', tar) where $v' \in O^+$, $tar \in Tar_i$, this means that the objects in string v' are placed in the region corresponding to tar.

6. $i_{out} \in \{0, 1, \ldots, m\}$ is the output region where 0 represents the environment.

In most P system models, rules are applied in a nondeterministic and maximally parallel manner. Nondeterminism implies that whenever more than one rule can possibly be applied to an object at a certain time step, then for each copy of the object, the system chooses a single rule to be applied on it. Maximal parallelism requires that all copies of objects that can evolve must evolve. In this paper, nondeterminism and maximal parallelism are applied in a restricted manner. We use a local parallelism called an exhaustive use of rules, similar to the application mode used in [6], where the rule nondeterministically chosen for an object in a region is applied to all copies of the object present in the region at that time step. The next definition describes this mode of application.

Definition 2. *An nTP system without dissolution with an exhaustive use of rules requires that all objects that can evolve must evolve and if more than one rule can possibly be applied on an object in a region, all copies of that object must evolve through a single nondeterministically chosen rule at a certain time step.*

Throughout this paper, rules for nTP systems run in a nondeterministic, maximally parallel mode with an exhaustive use of rules.

Next we define a numerical P system with migrating variables in a way similar to the definition found in [17].

Definition 3. *A Numerical P system with migrating variables (in short, MNP system) of degree* $m \geq 1$ *is a tuple of the form*

$$\Pi_{MNP} = (m, H, \mu, Var, (Pr_1, Var_1(0)), \ldots, (Pr_m, Var_m(0)), (x_{i_0}, h_0)),$$

- m is the number of membranes;
- H is an alphabet of labels for membranes in μ;
- μ is a hierarchical (cell-like) membrane structure with m membranes labeled with the elements in H;
- $Var = \{x_1, x_2, \ldots, x_n\}$ is a finite set of variables for the system;
- $Var_i(0)$, $1 \leq i \leq m$ is a vector of the form $(x_{i1}[a_{i1}], x_{i2}[a_{i2}], \ldots, x_{it_i}[a_{it_i}])$, which indicates that the variables initially present in region i are $x_{il} (\in Var)$ with value a_{il}, for $1 \leq l \leq t_i$;
- Pr_i, $1 \leq i \leq m$, is the finite set of programs in region i;
- $x_{i_0} \in Var, h_0 \in H$.

Each program $p_{j,i} \in Pr_i$, denoting the jth program in region i, is of the form

$$F_{j,i}(x_{p_1}, \ldots, x_{p_k}) \to c_{j,1}|(x_{r_1}, tar_1) + \cdots + c_{j,q}|(x_{r_q}, tar_q)$$

where $c_{j,1}, \ldots, c_{j,q}$ are natural numbers; $tar_1, \ldots, tar_q \in \{here, out, in\}$, the symbols $here, out, in$ are called *target commands*; all the variables $x_{p_1}, \ldots, x_{p_k}$ and $x_{r_1}, \ldots, x_{r_q}$ are from Var. $F_{j,i}(x_{p_1}, \ldots, x_{p_k})$ is the production function and $c_{j,1}|(x_{r_1}, tar_1) + \cdots + c_{j,q}|(x_{r_q}, tar_q)$ is the repartition protocol of the program. A program is said to be active at a particular time step if and only if all the variables $x_{p_1}, \ldots, x_{p_k}$ and $x_{r_1}, \ldots, x_{r_q}$ are present in membrane i where the program resides.

A program $p_{j,i}$ in region i is executed at a given step t in the following manner: The system first computes for the value of the production function $F_{j,i}$ from the current values of the variables in region i, taking note that the variables used in the production are reset to zero in the process. The production value is then distributed to variables $x_{r_1}, \cdots, x_{r_q}$ according to the repartition protocol $c_{j,1}|(x_{r_1}, tar_1) + \cdots + c_{j,q}|(x_{r_q}, tar_q)$. If we denote by $C_{j,i} = \sum_{s=1}^{q} c_{j,s}$ the sum of all coefficients of the repartition protocol, the value $q_{j_i} = F_{j,i}/C_{j,i}$ represents the "unitary portion" to be distributed to the variables $x_{r_1}, \ldots, x_{r_q}$. That is, for $1 \leq s \leq q$, $q_{j,i} \cdot c_{j,s}$ is added to x_{r_s} at time step $t+1$. In this work, we assume that $F_{j,i}$ is always divisible by $C_{j,i}$, a case which we will denote by *div*. Note that if this is not the case, then there are three possible decisions that the system can take: (i) the remainder is lost (production not immediately distributed is lost), (ii) the remainder is added to the production obtained at the next step (non-distributed production is carried over to the next step), and (iii) the system stops and no result is associated with that computation. These three cases are denoted by *lost*, *carry* and *stop*, respectively. If after applying all the rules a variable receives contributions from several programs, then all such contributions are summed to produce the next value of the variable. Each variable involved in the repartition protocol is then moved to the region as indicated by the target command associated with it. Specifically, *here* indicates that the variable remains in the same region i where the program is applied; *out* indicates that the variable will be moved to the region outside membrane i; *in* indicates that the variable will be moved to a child membrane of membrane i (in case there are several membranes inside membrane i, the variable is nondeterministically sent to only one of these membranes).

A computation of Π_{MNP} is described as follows: A configuration is a tuple representing the values of all the variables present in each membrane of the system at a given computation step. Initially, the variables in the system have the values specified in $Var_i(0), 1 \leq i \leq m$. A transition from a configuration at time t to a configuration at time $t+1$ is made by (a) choosing program(s) nondeterministically from each region according to a mode of application, (b) computing the value of the respective production function from the values of local variables at time t, and then (c) computing the values of variables at time $t+1$ as directed by repartition protocols. The execution is synchronized, that is, each of the mentioned three steps is done for all programs at the same time. A sequence of such transitions forms a computation. If no program in the system can be applied, we say that the system reaches a halting configuration. When the system halts, the value taken by the special variable x_{i0} in membrane h_0 is the number generated by the computation. if the variable x_{i0} does not appear in membrane h_0, then no result is associated with this computation.

Details on the modes of application and how the repartition of the program takes place are similar to that of traditional numerical P systems in [16,15]. We identify the three possible modes of application as follows: *sequential (seq)*, at each step, only one active program is chosen nondeterministically to be executed in every region; *one-parallel (oneP)*, for each region a maximal set of active programs is chosen nondeterministically to be executed such that no two programs have common variables in their production functions; *all-parallel (allP)*, all active programs are executed. Note that the *allP* mode of application indicates the deterministic parallel execution of programs in the system.

In [17], another method of applying programs was introduced called the *non-zero (NZ) assumption*: aside from the basic requirement that all variables involved in the program are present in the membrane, a program can be applied only when all the variables in the production have non-zero values.

Before we proceed to the next section, we define a notion of simulation in the context of P systems, as adapted from [8]. We note that a binary relation S over configurations in Π and Π' is a set of ordered pairs (c_1, c_2) where c_1 is a configuration in Π and c_2 is a configuration in Π'. The notation $c_1 S c_2$ indicates that element $(c_1, c_2) \in S$.

Definition 4. *Let* Π *and* Π' *be two P systems and let* S *be a binary relation over configurations in* Π *and* Π'. S *is a simulation over* Π *and* Π' *if whenever* $C_\Pi S C_{\Pi'}$, *if* $C_\Pi \Rightarrow C'_\Pi$, *there exists* $C'_{\Pi'}$ *such that* $C_{\Pi'} \overset{*}{\Rightarrow} C'_{\Pi'}$ *and* $C'_\Pi S C'_{\Pi'}$. *We say* $C_{\Pi'}$ *simulates* C_Π *and* Π' *simulates* Π.

In the succeeding section, we explore the relation of nTP systems without dissolution with an exhaustive use of rules having concentric membrane structure and MNP systems with respect to this notion of simulation.

3 Main Result

In this section, we show how we can construct an MNP system operating in oneP mode that simulates a given nTP system without dissolution with exhaustive use of rules having a concentric membrane structure. Before we formalize our simulation result, we note that for the nTP systems considered in this paper, no objects are transported to the environment and hence the environment cannot be an output region. Also, we allow the MNP system to have several output variables that must be contained in the same region.

We now present our main result in the following theorem:

Theorem 1. *For every nTP system* Π_{nTPe} *without dissolution with an exhaustive use of rules having a concentric membrane structure, there exists an MNP* *system* Π_{MNP} *with the NZ assumption operating in oneP mode that simulates* Π_{nTPe}.

Proof. To prove the theorem, we first construct an MNP system Π_{MNP} from a given nTP system Π_{nTPe}. Afterwards, we discuss how Π_{MNP} computes and we show that there is a simulation of Π_{nTPe} by Π_{MNP}. That is, in reference to Definition 4, we show that there exists a binary relation S over configurations in Π_{nTPe} and Π_{MNP} such that whenever $C_{\Pi_{nTPe}} S C_{\Pi_{MNP}}$, if $C_{\Pi_{nTPe}} \Rightarrow C'_{\Pi_{nTPe}}$, there exists $C'_{\Pi_{MNP}}$ such that $C_{\Pi_{MNP}} \overset{*}{\Rightarrow} C'_{\Pi_{MNP}}$ and $C'_{\Pi_{nTPe}} S C'_{\Pi_{MNP}}$.

Before we proceed, for ease of use in our discussion, we let Var_k denote the set of variables present in region k of an MNP system and we let O_k be the multiset of objects present in region k of an nTP system.

Now, given $\Pi_{\mathrm{nTPe}} = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_{out})$ without dissolution and with an exhaustive use of rules where $\mu = [_1[_2 \cdots [_{m-1}[_m]_m]_{m-1} \cdots]_2]_1$, consider

$$\Pi_{\mathrm{MNP}} = (m, \{1, 2, \ldots, m\}, \mu, Var, (Pr_1, Var_1(0)), \ldots, (Pr_m, Var_m(0)), (X, h_0))$$

where $X = \{v_{\alpha_{i_{out}}} \mid \alpha \in O\}$, $h_0 = i_{out}$ and

for every region k, $1 \le k \le m$,

- for every $a \in O$,
  - $v_{a_k} \in Var_k$, having initial value of $|w_k|_a$
  - $v'_{a_{k+1}} \in Var_k, 1 \le k \le m-1$ with initial values of 0
  - $v''_{a_{k-1}} \in Var_k, 2 \le k \le m$ with initial values of 0
  - $v'''_{a_k} \in Var_k$, with initial values of 0
- $g_k \in Var_k$
- for every evolution rule in R_k of the form $a \to w$, $a \in O$ and $w = (x, here)(y, out)(z, in)$ for some $x, y, z \in O^*$, if $x = x_1^{n_{x1}} x_2^{n_{x2}} \cdots x_{t_x}^{n_{xt_x}}$, $y = y_1^{n_{y1}} y_2^{n_{y2}} \cdots y_{t_y}^{n_{yt_y}}$ and $z = z_1^{n_{z1}} z_2^{n_{z2}} \cdots z_{t_z}^{n_{zt_z}}$ where $x_1, x_2, \ldots, x_{t_x}, y_1, y_2, \ldots, y_{t_y}, z_1, z_2, \ldots, z_{t_z} \in O$ and $n_{x_1}, n_{x_2}, \ldots, n_{x_{t_x}}, n_{y_1}, n_{y_2}, \ldots, n_{y_{t_y}}, n_{z_1}, n_{z_2}, \ldots, n_{z_{t_z}} \in \mathbb{N}$, then Pr_k contains the following program

$$n \cdot v_{a_k} \to \sum_{i=1}^{t_x} n_{xi} |(v'''_{x_{ik}}, here) + \sum_{i=1}^{t_y} n_{yi} |(v''_{y_{i(k-1)}}, out) + \sum_{i=1}^{t_z} n_{zi} |(v'_{x_{i(k+1)}}, in) \tag{1}$$

  with $n = \sum_{i=1}^{t_x} n_{xi} + \sum_{i=1}^{t_y} n_{yi} + \sum_{i=1}^{t_z} n_{zi} = |w|$
  (Note: In case $t_x + t_y + t_z = 1$, that is, $w = (b^n, tar)$ for some $b \in O$ and $tar \in \{in, out, here\}$, we may simply place either $n \cdot v_{a_k} \to 1|(v'_{b_{i+1}}, in)$ or $n \cdot v_{a_k} \to 1|(v''_{b_{i-1}}, out)$ or $n \cdot v_{a_k} \to 1|(v'''_{b_i}, here)$)
- for every $a \in O$, we include the following programs in Pr_k

$$2v'_{a_k} \to 1|(v_{a_k}, here) + 1|(v'_{a_k}, out), \qquad 2 \le k \le m \tag{2}$$

$$2v''_{a_k} \rightarrow 1|(v_{a_k}, here) + 1|(v''_{a_k}, in), \qquad 1 \leq k \leq m-1 \tag{3}$$

$$v'''_{a_k} \rightarrow 1|(v_{a_k}, here)), \qquad 1 \leq k \leq m \tag{4}$$

$$v'_{a_{k+1}} \rightarrow 1|(g_k, here), 1 \leq k \leq m-1; \;\; v''_{a_{k-1}} \rightarrow 1|(g_k, here), 2 \leq k \leq m \tag{5}$$

The system computes by doing the following steps in parallel in each region k:

1. Initially, since Π_{MNP} is working with NZ assumption, the only active programs are of the form in (1) where v_{a_k} have a nonzero initial value, for some $a \in O$. Since the system is working in *oneP* mode, if several of these programs in region k can possibly be applied on a variable, we nondeterministically select only one of these programs to be applied at this time step. The selected program place auxiliary variables $v'''_{a_k}, v''_{a_{k-1}}, v'_{a_{k+1}}$ to specified targets (here:region k, out : region $k-1$, or in : region $k+1$) and resets v_{a_k} to 0.
2. At the next time step, the values of the auxiliary variables of an object $b \in O$ received in a region k either from the outer region ($v'_{b_{k+1}}$) or from the inner region ($v''_{b_{k-1}}$) or from within region (v'''_{b_k}) are all summed up in the corresponding placeholder of the object (v_{b_k}) through the programs of the form in (2), (3) and (4) respectively. Through these same programs, all the auxiliary variables from outside the current region are returned to the regions they originated from while keeping their current values.

(*) Auxiliary variables that were used in the previous steps and returned to the region are reset to 0 by forwarding their values to the garbage variable g_x through programs of the form in (5). At the same time, if there are programs of the form in (1) that are applicable to the current values of the variables in Var_k, then the system goes back to step 1. Otherwise, the system halts.

Steps 1 and 2 correspond to the following scenario in Π_{nTPe}. An object a having multiplicity of $v_{a_k}(0)$ initially present in region k of Π_{nTPe} is evolved using the evolution rule $a \rightarrow w$ corresponding to the selected program in Pr_k. (Note that Π_{nTPe} runs with an exhaustive use of rules, hence only a single rule is applied on all copies of an object.) The evolved object is said to be consumed, and the objects produced in the evolution are placed in their specified targets: here - stays in region k, in - moved to inner region $k+1$, or out - moved to outer region $k-1$. In each region, the number of copies of an object b (which is the sum of unused copies of the object in the region and produced by evolution rules applied in the outer region and/or inner region and/or within the region) is equal to the value of v_{b_k}. When none of the evolution rules becomes applicable on the current multiset of objects found in each region, the system halts.

For the next part of our proof, we construct a binary relation S over configurations in nTPe and MNP as required by Definition 4. We say that $(C_{\mathrm{nTPe}}, C_{\mathrm{MNP}}) \in S$, if and only if $\forall a \in O_k$ if there are r copies of a in region k in C_{nTPe} then v_{a_k} has value r in C_{MNP}.

A transition $C_{\mathrm{nTPe}} \to C'_{\mathrm{nTPe}}$ involves, for at least one object a in some region k, applications of an evolution rule $a \to (x, \mathit{here})(y, \mathit{out})(z, \mathit{in})$ on all copies of a in region k (via an exhaustive use of rules). When such a rule is applied and suppose r is the multipicity of a in region k, then all r copies of a in region k are consumed, r copies of the multiset of objects in x are produced in region k, r copies of the multiset of objects in y are placed in region $k-1$ and r copies of the multiset of objects in z are placed in region $k+1$. Similarly, there exists a computation $C_{MNP} \overset{*}{\Rightarrow} C'_{MNP}$ where in C_{MNP}, a program in Pr_k is applied on v_{a_k} such that in C'_{MNP}, $r \cdot |x|_\alpha$ is added to v_{α_k} for each object α in x, $r \cdot |y|_\beta$ is added to $v_{\beta_{k-1}}$ for each object β in y, $r \cdot |z|_\gamma$ is added to $v_{\gamma_{k+1}}$ for each object γ in z and r is subtracted from v_{a_k}.

Hence, for every pair $(C_{\mathrm{nTPe}}, C_{\mathrm{MNP}}) \in S$ if there is a transition $C_{\mathrm{nTP}} \Rightarrow C'_{nTP}$, there exists a computation $C_{MNP} \overset{*}{\Rightarrow} C'_{MNP}$ where $(C'_{nTP}, C'_{MNP}) \in S$. The configuration C'_{MNP} is as described in the previous paragraph.

Note that from our description of how the system Π_{MNP} computes, we are assured that a computation step in Π_{nTPe} is performed in two time steps (steps 1 and 2) in Π_{MNP}. At the end of the computation, Π_{MNP} adds one more step (step (*)) for refreshing the values of the auxiliary variables before it halts.

□

We end this section with an illustration.

An Example. $\Pi_{\mathrm{nTPe}_1} = (\{b, d\}, [_1[_2[_3]_3]_2]_1, \lambda, bd^2, \lambda, R_1, R_2, R_3, 2)$ where $R_1 = \{d \to (b, \mathit{here})\}$, $R_2 = \{d \to d^2(b, \mathit{in}),\ d \to b^2(d, \mathit{out}),\ b \to (b, \mathit{in})\}$ and $R_3 = \{\}$.

A computation for Π_{nTPe_1} with an exhaustive use of rules is given below:

$C_0 : [_1\ [_2\ bd^2\ [_3\ ]_3\ ]_2\ ]_1$
$C_1 : [_1\ d^2[_2\ b^4\ [_3 b\ ]_3\ ]_2\ ]_1$
$C_2 : [_1\ b^2\ [_2\ [_3\ b^5\ ]_3\ ]_2\ ]_1$

This nTP system can be simulated by Π_{MNP_1} operating in oneP mode, where

$$\mathit{\Pi}_{MNP_1} = (3, \{1, 2, 3\}, [_1[_2[_3]_3]_2], \{v_{b_1}, v_{b_2}, v_{b_3}, v_{d_1}, v_{d_2}, v_{d_3}, v'_{b_2}, v'_{b_3}, v'_{d_2}, v'_{d_3},$$
$$v''_{b_1}, v''_{b_2}, v''_{d_1}, v''_{d_2}, v'''_{b_1}, v'''_{b_2}, v'''_{b_3}, v'''_{d_1}, v'''_{d_2}, v'''_{d_3}\},$$
$$(Pr_1, Var_1(0)), (Pr_2, Var_2(0)), (Pr_3, Var_3(0)), (\{v_{b_2}, v_{d_2}\}, 2)),$$

– $Var_1(0) = (v_{b_1}[0], v_{d_1}[0], v'_{b_2}[0], v'_{d_2}[0], v'''_{b_1}[0], v'''_{d_1}[0], g_1[0])$
– $Var_2(0) = (v_{b_2}[1], v_{d_2}[2], v'_{b_3}[0], v'_{d_3}[0], v''_{b_1}[0], v''_{d_1}[0], v'''_{b_2}[0], v'''_{d_2}[0], g_2[0])$
– $Var_3(0) = (v_{b_3}[0], v_{d_3}[0], v''_{b_2}[0], v''_{d_2}[0], v'''_{b_3}[0], v'''_{d_3}[0], g_3[0])$
– $Pr_1 = \{v_{d_1} \to 1|(v'''_{b_1}, \mathit{here})\} \cup \{2 \cdot v''_{b_1} \to 1|(v_{b_1}, \mathit{here}) + 1|(v''_{b_1}, \mathit{in}),$
$2 \cdot v''_{d_1} \to 1|(v_{d_1}, \mathit{here}) + 1|(v''_{d_1}, \mathit{in})\} \cup \{v'''_{b_1} \to 1|(v_{b_1}, \mathit{here}),$
$v'''_{d_1} \to 1|(v_{d_1}, \mathit{here})\} \cup \{v'_{b_2} \to 1|(g_1, \mathit{here}), v'_{d_2} \to 1|(g_1, \mathit{here})\}$
– $Pr_2 = \{3{\cdot}v_{d_2} \to 2|(v'''_{d_2}, \mathit{here}){+}1|(v'_{d_3}, \mathit{in}), 3{\cdot}v_{d_2} \to 2|(v'''_{b_2}, \mathit{here}){+}1|(v''_{d_1}, \mathit{out}),$
$v_{b_2} \to 1|(v'_{b_3}, \mathit{in})\} \cup \{2 \cdot v'_{b_2} \to 1|(v_{b_2}, \mathit{here}) + 1|(v'_{b_2}, \mathit{out}),$
$2 \cdot v'_{d_2} \to 1|(v_{d_2}, \mathit{here}) + 1|(v'_{d_2}, \mathit{out})\} \cup \{2 \cdot v''_{b_2} \to 1|(v_{b_2}, \mathit{here}) + 1|(v''_{b_2}, \mathit{in}),$
$2 \cdot v''_{d_2} \to 1|(v_{d_2}, \mathit{here}) + 1|(v''_{d_2}, \mathit{in}), v'''_{b_2} \to 1|(v_{b_2}, \mathit{here}), v'''_{d_2} \to 1|(v_{d_2}, \mathit{here}),$
$v''_{b_1} \to 1|(g_2, \mathit{here}), v''_{d_1} \to 1|(g_2, \mathit{here}), v'_{b_3} \to 1|(g_2, \mathit{here}), v'_{d_3} \to 1|(g_2, \mathit{here})\}$

– $Pr_3 = \{2{\cdot}v'_{b_3} \to 1|(v_{b_3}, here){+}1|(v'_{b_3}, out), 2{\cdot}v'_{d_3} \to 1|(v_{d_3}, here){+}1|(v'_{d_3}, out),$
$v'''_{b_3} \to 1|(v_{b_3}, here), v'''_{c_3} \to 1|(v_{c_3}, here), v'''_{d_3} \to 1|(v_{d_3}, here),$
$v''_{b_2} \to 1|(g_3, here), v''_{d_2} \to 1|(g_3, here)\}$

Note that at the halting configuration of a computation of Π_{MNP_1} simulating the above computation of Π_{nTPe_1}, aside from the garbage variables, the only nonzero variables are $v_{b_1} = 2$ and $v_{b_3} = 5$ which correspond to the objects b^2 in region 1 and b^5 in region 3 at the end of the computation of Π_{nTPe_1}.

We end this section with a corollary stating the amount of resources used in the above simulation.

Corollary 1. *Given an nTP system Π_{nTPe} as described in Theorem 1 having m membranes, n objects in O and r evolution rules running in t steps, there exists an MNP system Π_{MNP} with the NZ assumption in oneP mode simulating Π_{nTPe} in $2t+1$ steps having m membranes, $2n(m-1)$ migrating variables and $r + 4n(m-1) + nm$ programs.*

4 Final Remarks and Future Works

In this work, we presented a relation between nTP systems and MNP systems. Specifically, we provided a simulation of an nTP system with exhaustive use of rules having concentric membrane structure that runs in t steps on an MNP system operating in oneP mode running in $2t + 1$ steps. In this simulation, the multiplicities of the objects in the nTP system are the numerical values of the variables in the constructed MNP system.

The result in this paper is restricted to a class of nTP systems, that is, those that operate with exhaustive use of rules and having concentric membrane structure. For future work, we hope to extend this to other classes of nTP systems. Note that in nTP systems, evolution rules with target command in_j indicates a specific inner membrane as its target. In [13], a general definition of transition P systems uses the target command in (instead if in_j) allowing a degree of nondeterminism in the case where there are several inner membranes. Since in MNP systems, programs with target command in in the repartition protocol nondeterministically selects an inner membrane as the target, if the target command in is used for the nTP systems, then the result here can be easily extended. In this work, we restricted the parallelism of the nTP system in the application of rules through an exhaustive use of rules. If such restriction is not in place, can such a system be simulated by some MNP system? What if there are cooperative rules in the system? How about simulations of MNP systems by nTP systems?

Finally, a more promising application of simulations of P systems on other P systems is on distributed P (dP) systems. In dP systems we have a dP scheme composed of several P systems, possibly of different types, interacting with each other [14]. If one of these components becomes nonoperational (due to some reason not important in this work), can we replace it by another system (from a different family, perhaps) that interacts with the other components in the same way the original system works? If simulation relations between different P systems are available then such substitution can be easily done.

Acknowledgments

N.H. S. Hernandez is supported by the Vea Family Technology for All professorial chair. H.N. Adorna is funded by a DOST-ERDT research grant and the Semirara Mining Corporation professorial chair of the UPD CoE.

References

1. Cabarle, F.G.C., Adorna, H.N.: On Structures and Behaviors of Spiking Neural P Systems and Petri Nets. E. Csuhaj-Varjú et al. (Eds.): CMC 2012, LNCS 7762, pp. 145-160, Springer-Verlag (2013)
2. Ciobanu, G., Păun, Gh., Pérez-Jiménez, M.J.: Chapter 1 Introduction to Membrane Computing. Applications of Membrane Computing, Springer-Verlag (2006)
3. Frisco, P.: P Systems, Petri Nets, and Program Machines. WMC 2005, LNCS 3850, pp. 209-223, (2006)
4. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Computing Backwards with P systems. WMC10, Curtea de Argeş, Romania, pp. 282-295, (2009)
5. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979)
6. Ionescu, M., Paun G., Yokomori T.: Spiking neural P systems with an exhaustive use of rules. International Journal of Unconventional Computing. 3(2), pp. 135-153 (2007)
7. Juayong R.A.B., Hernandez N.H.S., Cabarle F.G.C., Adorna H.N.: Relating Transition P Systems and Spiking Neural P Systems. In: Mauri G., Dennunzio A., Manzoni L., Porreca A.E. (eds) Unconventional Computation and Natural Computation. UCNC 2013. Lecture Notes in Computer Science, vol 7956. Springer, Berlin, Heidelberg (2013)
8. Milner, R.: Communicating and Mobile Systems: the Pi-Calculus. Cambridge University Press (1999)
9. Păun, G.: Computing with Membranes. Journal of Computer and System Sciences vol. 61, pp.108-143 (2000)
10. Păun, G., Rozenberg, G., Salomaa A.: The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
11. Păun, G.: Membrane Computing - An Introduction. Springer-Verlag, Berlin (2002)
12. Păun, G.: Membrane Computing: History and Brief Introduction. In: Fundamental Concepts in Computer Science and Engineering: Texts, Volume 3, Gelenbe, E, Kahane, J. (eds.), World Scientific, pp. 17-41 (2009)
13. Păun, G.: Introduction to Membrane Computing. In: Applications of Membrane Computing, Natural Computing Series. Ciobanu, G., Pérez-Jiménez, M., Păun, G. (eds.), Springer Science and Business Media, pp. 1-42 (2007)
14. Păun, G., Pérez-Jiménez, M.: Solving Problems in a Distributed Way in Membrane Computing: dP Systems. International Journal of Computers Communications & Control, 5(2), pp. 238-250 (2010)
15. Păun, G. and Păun, R.: Membrane computing and economics: Numerical P Systems. Fundamenta Informaticae, 73(1), pp. 213-227 (2006)
16. Vasile, C., Pavel, A. and Dumitrache, I.: Universality of Enzymatic Numerical P systems. International Journal of Computer Mathematics. Vol.90, No.4, pp. 869-879 (2013)
17. Zhang, Z., Wu, T., Păun, A., and Pan, L.: Numerical P systems with migrating variables. Theoretical Computer Science, Volume 641, pp. 85-108 (2016)

Extended Abstracts

Cellular P system-based Modeling Framework for Cooperative Performance on High-tech Enterprises' Innovation (Extended Abstract)

Ping Chen and Xiyu Liu

School of Management Science and Engineering, Shandong Normal University
{hxscp444, sdxyliu}@163.com

Abstract: Cooperative innovation is important under the fierce competition in the market environment. Different from the previous streamlined or ring model for cooperative innovation, we propose a cellular model on cooperative innovation for high-tech enterprises. The model is analyzed by analytic hierarchy process and is simplified based on cellular P system. The model contains two layers, one is the performance of cooperative innovation and another layer consists of human resources, energy resources and technology innovation project. It demonstrates the inclusion relationship between two layers, and also shows the interaction of three main factors for cooperative innovation at the second layer. In the work, a case of high-tech enterprises in Shandong, China is studied. As a result, the study of the case shows the accurate of the model we proposed.

Keywords: AHP, cellular P system, cooperative innovation, high-tech

1 Modeling framework

The model proposed in this paper is based on AHP and combined with the structural model of the cellular P system.

The model is modified by the common structure of cellular P systems. The membrane structure of the cellular P system is mainly composed of the main membrane (also used to be called skin membrane) and the other membrane arranged in a hierarchical structure within the skin membrane. The cooperative innovation model has a total of two layers with four membranes, including skin membrane which restricts the region of the cooperative innovation performance and three elementary membranes, which prescribe a limit to the region of human resources, technology projects and energy resources. Human resources, technology innovation project and energy resource are regarded as the input which play an important role in the performance of cooperative innovation. As shown simply in figure 1, the appropriate input of human resource, technology innovation project and energy resource have a positive impact on the performance cooperative innovation.

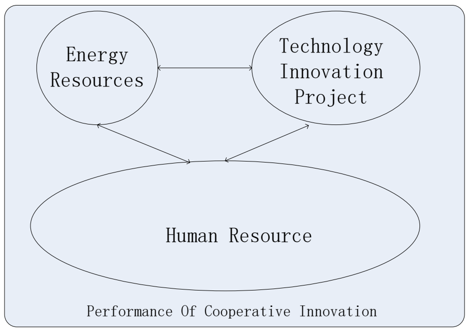

Energy Resources
Technology Innovation Project
Human Resource
Performance Of Cooperative Innovation

Fig. 1. Model on cooperative innovation proposed in this paper is based on analytic hierarchy process and the cellular P system

During the process of framework modeling, we use analytic hierarchy process the give its validation. The first step in using AHP is to develop the hierarchy by breaking down the problem into its components. As we have assumed, the objectives for the goal include three main factors, which are human resources, technological innovation project and energy resources and the alternatives are original energy resources, environment protection resources, technical reconstruction, patents, technical craft workers, the staff of bachelor or above and administrative staff as shown in figure 2.

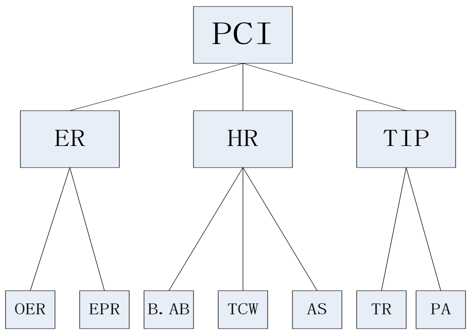


Fig. 2. Three layers of cooperative innovation include their factors. The factors are PCI (performance of cooperative innovation), ER (energy resources) containing OER(original energy resources) and EPR(environment protection resources), HR (human resources) including B.AB (staff with bachelor or above), TCW (technical craft workers) and AS(administrative staff) and TIP (technology innovation projects) which contains technology reconstruction (TR) and patents (PA)

2 Validation and the case study in Shandong, China

Shandong Province, the population ranked second in China, the overall economic strength ranked third, is of great significance to China's development. Enterprises in Shandong Province play a critical role on the economic development, while high-tech enterprises matter. And it is also meaningful to verify the model of cooperative innovation based on the development of Shandongs high-tech enterprises. The data is collected on high-tech enterprises about their financial input and output on innovation and development during the three-year period 2011 to 2013.The total number of enterprises participating in the survey was 2000, and their development mainly based on innovation according to the standard of high-tech industry and the development data.Using these enterprises financial input and output data, our goal is to verify that the model is in line with the reality of enterprises development. In the 2000 companies, we conducted a simple random sampling of sampling with replacement, obtaining three groups may contain duplication of business development economic data samples, and each sample contains 300 enterprises. Then, we extracted the sample of the three years of data analysis to see if the cooperative innovation development mode is consistent with the model we put forward.

Through the analysis of its input, its output and their relationship of the enterprise for three consecutive years, we believe that our model is 100% consistent with the high-tech enterprise collaborative innovation development model.

3 Conclusions

As a conclusion, this study developed a framework for modeling the cooperative innovation process of high-tech enterprises. A case study of 2000 high-tech enterprises in Shandong, China is used to illustrate the model. The results show that the model we proposed in line with the reality of high-tech enterprises development. Some main factors have positive effect on the cooperative innovation. Meanwhile, they play a role with each other to push the process of cooperative innovation. According to the case study, the framework can be applied to high-tech enterprises in general and the list of factors are facts. Through analytic hierarchy process, the weight of factors simply changes and the value of different factors is different. Finally, future work should attempt to distinguish between different industries and consider their weight among various factors.

Acknowledgments

This work was supported by the Natural Science Foundation of China (No.61472231), Humanities and Social Sciences Project of Ministry of Education, China (No.12YJA630152), Social Science Fund of Shandong Province, China (No.11CGLJ22), Social Science Fund of Shandong Province, China (No.16BGLJ06), Natural Science Foundation of China(No.61502283), Natural Science Foundation of China(No.61640201).

References

1. Gunday, Gurhan, et al., Effects of innovation types on firm performance., International Journal of Production Economics, 133.2, 662-676 (2011)
2. Rejeb, Helmi Ben, et al., Measuring innovation best practices: Improvement of an innovation index integrating threshold and synergy effects , Technovation, 28.12, 838-854 (2008)
3. Muller, A, L. Valikangas, and P. Merlyn, Metrics for innovations: Guidelines for developing a customized suite of innovation metrics , Strategy & Leadership, 33.1, 66-66 (2005)
4. Nambisan, Satish, Software firm evolution and innovationCorientation , Journal of Engineering & Technology Management, 19.2, 141-165 (2002)
5. Khurum, Mahvish, S. Fricker, and T. Gorschek, The contextual nature of innovation C An empirical investigation of three software intensive products , Information & Software Technology , 57.1, 595-613 (2015)
6. Zhang, Xinyue, and W. Song, Structure, Evolution and Hot Spots of Cooperation Innovation Knowledge Network , Open Journal of Applied Sciences, 5.4, 121-134 (2015)
7. Weiers, Georg, Innovation Through Cooperation , Management for Professionals, (2014)
8. Savin, Ivan, and A. Egbetokun, Emergence of innovation networks from R&D cooperation with endogenous absorptive capacity , Ssrn Electronic Journal, 64, 82-103 (2013)
9. Yip, George, and B. Mckern, Innovation in emerging markets C the case of China , International Journal of Emerging Markets, 9.1, 2-10(9) (2014)
10. Farok J. Contractor, Punching above their weight: The sources of competitive advantage for emerging market multinationals , International Journal of Emerging Markets, 8.4, (2013)
11. Liu, Xiyu, J. Xue, and X. Yu, A muti-agent membrane computing technique for conceptual design , IEEE International Conference on Computer Supported Cooperative Work in Design, 133-138 (2013)

Generalized P Colony Automata and Their Relation to P automata [⋆] (Extended Abstract)

Kristóf Kántor, György Vaszil

Department of Computer Science, Faculty of Informatics
University of Debrecen
Kassai út 26, 4028 Debrecen, Hungary
{kantor.kristof, vaszil.gyorgy}@inf.unideb.hu

1 Introduction

P colonies are variants of very simple membrane systems, which resemble the so-called colonies of grammars, see [7], that is, collections of very simple generative grammars which work in cooperation, and together they are able to generate fairly complicated languages.

P colonies also consist of a collection of very simple computing agents which interact in a shared environment, see [8, 9]. The agents and the environment described by multisets of objects which are processed by rules enabling the transformation of the objects and the exchange of objects between the colony members and the environment. The rules are grouped into programs, and a computation consists of a sequence of computational steps during which the colony members execute their programs in parallel, until the system reaches a halting configuration.

P colony automata, a variant of P colonies characterizing string languages were introduced in [2], the variants called generalized P colony automata were introduced in [5]. One of the motivations of introducing generalized P colony automata was to make the model resemble more to P automata, which was introduced in [4]. In the case of generalized P colony automata, the computation of the colony defines an accepted multiset sequence, which is turned into a set of accepted string by a non-erasing mapping (as in P automata).

2 P Automata and Generalized P Colony Automata

A *genPCol automaton* of capacity k and with n cells, $k, n \geq 1$, is a construct $\Pi = (V, e, w_E, (w_1, P_1), \ldots, (w_n, P_n), F)$ where

- V is the *alphabet* of the automaton, its elements are called *objects*;
- $e \in V$ is the *environmental object* of the automaton;

[⋆] Research supported in part by grant no. MAT120558 of the National Research, Development and Innovation Office, Hungary.

– $w_E \in (V - \{e\})^*$ is a string representing the multiset of objects different from e which is found in the environment initially;
– $(w_i, P_i), 1 \leq i \leq n$, specifies the i-th *cell* where w_i is a multiset over V, it determines the initial contents of the cell, and its cardinality $|w_i| = k$ is called the *capacity* of the system. P_i is a set of *programs*, each program is formed from k rules of the following types (where $a, b \in V$):
  - *tape rules* of the form $a \overset{T}{\rightarrow} b$, or $a \overset{T}{\leftrightarrow} b$, called rewriting tape rules and communication tape rules, respectively; or
  - *nontape rules* of the form $a \rightarrow b$, or $a \leftrightarrow b$, called rewriting (nontape) rules and communication (nontape) rules, respectively.

  A program is called a *tape program* if it contains at least one tape rule. (The names "tape" rule and "tape" program are motivated by the effect of the use of these rules/programs: as "ordinary" automata read symbols by processing an input tape, P colony automata read symbols by applying these rules/programs. See below for more details.)
– F is a set of *accepting configurations* of the automaton which we will specify in more detail below.

A genPCol automaton reads an input word during a computation. A part of the input (possibly consisting of more than one symbol) is read during each configuration change: the processed part of the input corresponds to the multiset of symbols introduced by the tape rules of the system.

A *configuration* of a genPCol automaton is an $(n+1)$-tuple $(u_E, u_1, \ldots, u_n)$, where $u_E \in (V - \{e\})^*$ represents the multiset of objects different from e in the environment, and $u_i \in V^*,\ 1 \leq i \leq n$, represents the contents of the i-th cell. The *initial configuration* is given by $(w_E, w_1, \ldots, w_n)$, the initial contents of the environment and the cells. The elements of the set F of *accepting configurations* are given as configurations of the form $(v_E, v_1, \ldots, v_n)$, where

– $v_E \subseteq (V - \{e\})^*$ represents a multiset of objects different from e being in the environment, and each
– $v_i \in V^*,\ 1 \leq i \leq n$, is the contents of the i-th cell.

Instead of the different computational modes used in [2], in genPCol automata, we apply the programs in the maximally parallel way, that is, in each computational step, every component cell non-deterministically applies one of its applicable programs. Then we collect all the symbols that the tape rules "read" (these multisets are denoted by $read(p)$ for a program p in the definition below), this is the multiset read by the system in the given computational step. A successful computation defines in this way an accepted sequence of multisets: the sequence of multisets entering the system during the steps of the computation.

Let $\Pi = (V, e, w_E, (w_1, P_1), \ldots, (w_n, P_n), F)$ be a genPCol automaton. The *set of input sequences accepted by* Π is defined as

$$\begin{aligned} A(\Pi) = \{u_1 u_2 \ldots u_s \mid\ & u_i \in (V - \{e\})^*,\ 1 \leq i \leq s, \text{ and there is a configuration} \\ & \text{sequence } c_0, \ldots, c_s, \text{ with } c_0 = (w_E, w_1, \ldots, w_n),\ c_s \in F, \text{ and} \\ & c_i \Longrightarrow c_{i+1} \text{ with } \bigcup_{p \in P_{c_i}} read(p) = u_{i+1} \text{ for all } 0 \leq i \leq s-1\}. \end{aligned}$$

Let Π be a genPCol automaton, and let $f : (V - \{e\})^* \to 2^{\Sigma^*}$ be a mapping, such that $f(u) = \{\varepsilon\}$ if and only if u is the empty multiset.

The *language accepted by* Π with respect to f is defined as

$$L(\Pi, f) = \{f(u_1)f(u_2)\dots f(u_s) \in \Sigma^* \mid\ u_1u_2\dots u_s \in A(\Pi)\}.$$

We define the following language classes.

- $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{com-tape}(k))$ is the class of languages accepted by generalized PCol automata with capacity k and with mappings from the class $\mathcal{F}$ where all the communication rules are tape rules,
- $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{all-tape}(k))$ is the class of languages accepted by generalized PCol automata with capacity k and with mappings from the class $\mathcal{F}$ where all the programs must have at least one tape rule,
- $\mathcal{L}(\text{genPCol}, \mathcal{F}, *(k))$ is the class of languages accepted by generalized PCol automata with capacity k and with mappings from the class $\mathcal{F}$ where programs with any kinds of rules are allowed.

Let $f : V^* \to 2^{\Sigma^*}$, for some alphabets V and Σ, and let the mapping f_{perm} and the class of mappings TRANS be defined as follows:

- $f = f_{perm}$ if and only if $V = \Sigma$ and for all $v \in V^{(*)}$, we have $f(v) = \{a_1a_2\dots a_s \mid |v| = s, \text{ and } a_1a_2\dots a_s \text{ is a permutation of the elements of } v\}$;
- $f \in$ TRANS if and only if for any $v \in V^{(*)}$, we have $f(v) = \{w\}$ for some $w \in \Sigma^*$ which is obtained by applying a finite transducer to the string representation of the multiset v, (as w is unique, the transducer must be constructed in such a way that all string representations of the multiset v as input result in the same $w \in \Sigma^*$ as output, and moreover, as f should be nonerasing, the transducer produces a result with $w \neq \varepsilon$ for any nonempty input).

We denote the above defined language classes as $\mathcal{L}_X(\text{genPCol}, Y(k))$, where $X \in \{f_{perm}, \text{TRANS}\},\ Y \in \{\text{com-tape}, \text{all-tape}, *\}$.

3 New Results on Systems with Input Mappings from TRANS

For any class of mappings $\mathcal{F}$, we have (see [6])

1. $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{com-tape}(k)) \subseteq \mathcal{L}(\text{genPCol}, \mathcal{F}, *(k))$ and $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{all-tape}(k)) \subseteq \mathcal{L}(\text{genPCol}, \mathcal{F}, *(k)$ for any $k \geq 1$; and
2. $\mathcal{L}(\text{genPCol}, \mathcal{F}, X(k)) \subseteq \mathcal{L}(\text{genPCol}, \mathcal{F}, X(k+1))$ for any $k \geq 1$ and $X \in \{\text{com-tape}, \text{all-tape}, *\}$.

The computational capacity of genPCol automata with input mappimg f_{perm} was investigated in [5] and [6]. It was shown that $\mathcal{L}_{perm}(\text{genPCol}, *(1)) = \mathcal{L}(RE)$, thus, it is not surprising, but the same holds also for the class of mappings TRANS.

Proposition 1.

$$\mathcal{L}_{\text{TRANS}}(\text{genPCol}, *(1)) = \mathcal{L}(RE).$$

A similar result holds for all-tape systems with capacity at least two. From [6] we have that $\mathcal{L}_{perm}(\text{genPCol}, \text{all-tape}(k)) = \mathcal{L}(\text{RE})$ for $k \geq 2$, and we can show the same for systems with input mappings from TRANS.

Proposition 2.

$$\mathcal{L}_{\text{TRANS}}(\text{genPCol}, \text{all-tape}(k)) = \mathcal{L}(\text{RE}) \textit{ for } k \geq 2.$$

For systems with capacity one, it is not difficult to see that all regular langugaes can be characterized, but a more precise characterization of the corresponding langugae classes are still missing.

Proposition 3.

$$\text{REG} \subseteq \mathcal{L}_{\text{TRANS}}(\text{genPCol}, \text{X}(1)), \textit{ for } \text{X} \in \{\text{all-tape}, \text{com-tape}\}.$$

The characterization of langugaes of com-tape systems is an interesting research direction. Similarly to systems with input mapping f_{perm}, we have the following, where r-1LOGSPACE denotes the class of languages characterized by so-called *restricted one-way logarithmic space* bounded Turing machines, see [3] for more on this complexity class.

Proposition 4.

$$\mathcal{L}_{\text{TRANS}}(\text{genPCol}, \text{com-tape}(2)) \subseteq \text{r-1LOGSPACE}.$$

As the class of languages characterized by P automata is strictly included in r-1LOGSPACE, the above statement does not give any information on the relationship of the power of P automata and genPCol automata. We know, however (see [6]), that genPCol automata with f_{perm} and com-tape programs can characterize languages that cannot be accepted by P automata using the mapping f_{perm}.

As P automata with sequential rule application and input mappings from TRANS characterize exctly the language class r-1LOGSPACE, the relationship of this language class and genPCol automata with input mappings from TRANS seems to be an especially interesting research direction.

Further, the effect of using *checking rules*, as defined in [8] for P colonies, is also an interesting topic for further investigations, just as the investigation of systems with other classes of input mappings besides f_{perm}.

References

1. L. Ciencialová, E. Csuhaj-Varjú, A. Kelemenová, Gy. Vaszil, Variants of P colonies with very simple cell structure. *International Journal of Computers Communication and Control*, **4** (2009), 224–233.

2. L. Cienciala, L. Ciencialová, E. Csuhaj-Varjú, Gy. Vaszil, *PCol automata: Recognizing strings with P colonies.* In: M. A. Martínez del Amor, Gh. Păun, I. Pérez Hurtado, A. Riscos Nuñez (eds.), Eighth Brainstorming Week on Membrane Computing, Sevilla, February 1-5, 2010, Fénix Editora, 2010, 65–76.
3. E. Csuhaj-Varjú, M. Oswald, Gy. Vaszil, P automata. In [10], chapter 6, 144–167.
4. E. Csuhaj-Varjú, Gy. Vaszil, P automata or purely communicating accepting P systems. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (eds.), *Membrane Computing. International Worskhop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002, Revised Papers.* LNCS 2597, Springer Berlin Heidelberg, 2003, 219–233.
5. K. Kántor and Gy. Vaszil Generalized P Colony Automata *Journal of Automata, Languages and Combinatorics* **19** (2014), 145–156.
6. K. Kántor and Gy. Vaszil On the Class of Languages Characterized by Generalized P Colony Automata *Accepted*.
7. J. Kelemen, A. Kelemenová, A grammar-theoretic treatment of multiagent systems. *Cybernetics and Systems*, **23** (1992), 621–633.
8. J. Kelemen, A. Kelemenová, Gh. Păun, Preview of P colonies: A biochemically inspired computing model. In: M. Bedau et al. (eds.), *Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX).* Boston Mass., 2004, 82–86.
9. A. Kelemenová, P Colonies. In [10], chapter 23.1, 584–593.
10. Gh. Păun, G. Rozenberg, A. Salomaa, editors, *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.

A Novel Membrane System with CPSO Algorithm for Clustering Problem (Extended Abstract)

Lin Wang[1] and Xiyu Liu[1,*]

School of Management Science and Engineering, Shandong Normal University,Jinan, China
moanfly@163.com, sdxyliu@163.com,

Abstract: This paper focus on clustering problem and proposed a membrane system which combined the Particle Swarm Optimization (PSO) algorithm, new evolutionary and communication rules are described . In this mechanism, Each particle represent one object, and a novel local search strategy is presented with logistic method which used to adjust parameters, a Dynamic Disturbance Term (DDT) was added in the velocity updating formula of the Chaotic PSO (CPSO) algorithm. At last, we apply the new algorithm on some benchmark clustering problems, the experiment validated the effectiveness and efficiency of the P-System CPSO (P-PSO) algorithm.
Keywords: Membrane Computing; PSO algorithm; Chaotic Map; Dynamic Disturbance Term.

1 Introduction

Data Clustering is one of the most popular data mining techniques, It aims to pack similar data into groups based on the characteristics of the data and discover useful information form data [1-4]. In this paper, we propose a new algorithm for clustering, which is the combination of Membrane Computing and PSO algorithm. A Dynamic Disturbance Term (DDT)[5] was added in the velocity updating formula. Also the membrane structure is constructed to resolve the clustering problem, new rules and membranes structure are described, and the skin membrane output the best object. Finally, we apply the new algorithm in the benchmark clustering problem, the experiment shows the P-CPSO has a good performance than other algorithms.

2 CPSO Membranes Clustering Algorithm (P-CPSO)

2.1 The CPSO Clustering Algorithm

We put forward a method based on dynamic disturbance term (DDT), a successively DDT was added into the velocity updating formula of the particle. it takes the following formula [6]:

$v_{ij}(t+1) = w * v_{ij}(t) + c_1 * r_1 * (p_{ij}(t) - x_{ij}(t)) + c_2 * r_2 * (p_{gj}(t) - x_{ij}(t)) + M;$ (1)
$M = k * (t/T_{max} - 0.5);$ (2)
K: accommodation coefficient which the value between [0,1]. And we use the logistic map to adjust the parameters, The parameters r_1, r_2 are self-adaptively adjusted by means of chaos strategy[7].

2.2 A Cell-like P System Designed for CPSO Algorithm

2.2.1 Membrane Structure

This paper uses the Cell-like P system, Fig 1 gives a initial cell-like membrane system, in which its 5 membranes are arranged as a hierarchical structure, which has symporter/inverse transfer rules.

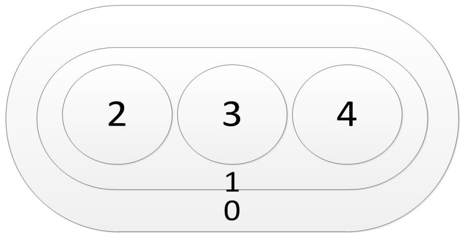


Fig. 1. Membrane structure of P-CPSO algorithm

2.2.2 Three Rules of P-System

Evolutionary Rules: In the elementary membranes 2-4, The best objects is described as:$C_{lbestj}^{(i+1)} \doteq C_{lbest}^{(i+1)}, iff(C_{lbest}^{(i+1)}) < f(C_{lbest}^{i})$Otherwise $C_{lbestj}^{(i+1)} \doteq C_{lbest}^{i}$, Where $C_{lbest}^{(i+1)}$ is the best object found in j elementary membrane on the i+1th iteration. CPSO-rules derives from the idea of CPSO algorithm.
Selection Rules: In the membrane 1, we take the lowest fitness value of the objects in all elementary membrane 2-4 as the global extreme in the membrane 1. The selection strategy is described as follow:$C_{gbest}^{i} \doteq \min C_{lbestj}^{i}, j \doteq 1,2,3$;$C_{gbest}^{(i+1)} \doteq C_{gbest}^{(i+1)}, iff(C_{gbest}^{(i+1)}) \leq f(C_{gbest}^{i})$;Otherwise $C_{gbest}^{(i+1)} \doteq C_{gbest}^{i}$;Where C_{gbest}^{i} represent the best objects in three elementary membranes 2-4.
Communication Rules: In each generation, for the elementary membrane 2-4, it select 3 optimal objects $C_{lbest}^{i}$and transfer them to the membrane 1. The membrane 1 select one optimal objectsC_{gbest}^{i} from the 3 objects C_{lbestj}^{i} according to the formula. And transfer the best objects C_{gbest}^{i} to the skin membrane 0. Thus, the skin membrane always holds an object. Which is the optimal object to data.

2.3 The Fitness of Objects

The objects evaluation criterion criteria are calculated as follows:$D(c_1, c_2, \cdots, c_k) = \sum_{(i=1)}^{k} \sum_{(a_j \in c_i)} (a_j - c_i)^2$;$F(C_i) = 1/D$; Where $a_1, a_2, \cdots a_n$is the data points of the corresponding cluster , n is the number of the data points.$\|a_p - c_m\| < \|a_p - c_t\|$; $p = 1, 2, 3, \cdots, n$; $t = 1, 2, 3, \cdots, k$; $m = 1, 2, 3, \cdots, k$;where $t \neq m$;where n is the number of the data points and k is the number of clusters.

3 Experimental Results

In this section, We take some experimental data for this algorithm. These data has three dimensions on the search space, and the datasets consists 51 data which was the experiment objects,The correct clustering data is :2; 3; 8; 9; 10; 13; 14; 17; 21; 23; 26; 27; 29; 31; 39; 40; 43; 46; 49; 51; 4; 5; 7; 11; 12; 20; 25; 28; 32; 36; 37; 38; 41; 44; 45; 50;1; 18; 22; 30; 34; 42; 47; 48; The swarm population size is 100. $c_1 \doteq c_2 \doteq 2$. $w_{min} = 0.4$, $w_{max} = 1.25$. And the correct clusters data on the Table.1.

Table 1. The results of the P-CPSO clustering algorithm

Cluster	Data points	Clustering
P-CPSO	C_1	2; 3; 8; 9; 10; 13; 14; 17; 21; 23; 26; 27; 29; 31; 39; 40; 43; 46; 49; 51
P-CPSO	C_2	6; 15; 16; 19; 24; 33; 35
P-CPSO	C_3	1; 18; 22; 30; 34; 42; 47; 48
P-CPSO	C_4	4; 5; 7; 11; 12; 20; 25; 28; 32; 36; 37; 38; 41; 44; 45;50

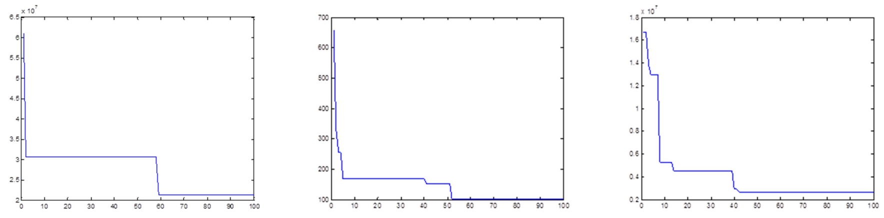


Fig. 2. Convergence of the P-CPSO algorithm (Iris, Wine)

At last, we take the instances of the Iris, Wine from the UCI dataset to solve the clustering problem. From the Figure 2 and Table 1, Table 2, it is possible to see the performance of P-CPSO algorithm is improved by incorporating the membrane system and logistic search method .

Table 2. Results of clustering problem for the P-CPSO algorithm

Clusters	The number of clusters	Computation Time Time	Beat Optimal Values
Data	4	7.5241	$2.1277 * 10^7$
Iris	3	8.7795	$1.0272 * 10^2$
Wine	3	10.8335	$0.2662 * 10^7$

4 Conclusions

This paper has proposed and tested a new clustering method, P-CPSO algorithm is able to solve the data clustering problem, which combine the membrane computing with the PSO algorithm. The new algorithm applied in the benchmark clustering problem, and proved its effectiveness. Because large problems is NP-hard problem in a linear or polynomial time. However, this paper has the limit of space consideration. Therefore, our future work is to study the other swarm algorithm combine with membrane computing. And used the membrane system with active membranes to solve the other combination problems in a linear time.

Acknowledgment. Project supported by National Natural Science Foundation of China (61472231,61502283,61640201), Ministry of Education of Humanities and Social Science Research Project, China (12YJA630152), Social Science Fund Project of Shandong Province, China (16BGLJ06,11CGLJ22).

References

[1]Zhao, Yuzhen, X. Liu, and X. Yan. "A Grid-Based Chameleon Algorithm Based on the Tissue-Like P System with Promoters and Inhibitors.*Journal of Computational Theoretical Nanoscience* 13.6(2016):3652-3658.
[2]Paun, Gheorghe. "Computing with Membranes. *Journal of Computer Sytem Sciences* 61.1(2000):108-143.
[3]Xue, Jie, and X. Liu. "Lattice based communication P systems with applications in cluster analysis." *Soft Computing* 18.7(2014):1425-1440.
[4]Peng, Hong, et al. "An automatic clustering algorithm inspired by membrane computing." *Pattern Recognition Letters* 68(2015):34-40.
[5] Zhang, Yaqiong, J. Lin, and H. Zhang. "A Hierarchical Teaching Mode of College Computer Basic Application Course Based on K-means and Improved PSO Algorithm." 11.10(2016):53.
[6]Eberhart, R., and J. Kennedy. "A new optimizer using particle swarm theory." *International Symposium on MICRO Machine and Human Science IEEE*, 2002:39-43.
[7]Wu, D. H., et al. "Adaptive double particle swarms optimization algorithm based on chaotic mutation." *Control Decision* 7331.1(2011):148-155.

Author Index