International Workshop on Computing with Biomolecules

books@ocg.at
BAND 244

Gedruckt mit Förderung des Bundesministeriums
für Wissenschaft und Forschung in Wien.

Erzsébet Csuhaj-Varjú, Rudolf Freund,
Marion Oswald, and Kai Salomaa (eds.)

International Workshop on Computing with Biomolecules

August 27th, 2008
Wien, Austria

Druck: Druckerei Riegelnik
1080 Wien, Piaristengasse 19

ISBN   978-3-85403-244-1

Preface

The International Workshop on *Computing with Biomolecules* took place in Vienna on August 27th, 2008, in conjunction with the Seventh International Conference on Unconventional Computation, UC 2008, which was held during the period August 25th to August 28th, 2008.

The venue for the workshop was the Parkhotel Schönbrunn in immediate vicinity of Schönbrunn Palace, with its beautiful buildings and wide parks being one of the most important cultural monuments in Austria. Vienna, located in the heart of central Europe, still reflects its former outstanding historical role as the capital of a great empire and the residence of the Habsburgs in its architectural monuments, its famous art collections and its rich cultural life, in which music has always played an important part.

Whereas the International Conference on Unconventional Computation UC 2008, see `http://www.emcc.at/UC2008`, was devoted to many aspects of unconventional computation, especially to quantum computing and molecular computing, the main focus of the International Workshop on Computing with Biomolecules, see `http://www.emcc.at/CBM2008/`, was molecular computing with topics mainly related to membrane systems.

The workshop was organized by Erzsébet Csuhaj-Varjú (Hungarian Academy of Sciences, Hungary) and Rudolf Freund (Vienna University of Technology, Austria); the Programme Committee consisted of

– Artiom Alhazov, Åbo Akademi University, Finland,
– Daniela Besozzi, Università degli Studi di Milano, Italy,
– Erzsébet Csuhaj-Varjú, Hungarian Academy of Sciences (co-chair),
– Giuditta Franco, Università degli Studi di Verona, Italy,
– Rudolf Freund, Vienna University of Technology, Austria (co-chair),
– Marion Oswald, Vienna University of Technology, Austria,
– Francisco J. Romero-Campero, University of Nottingham, UK, and Universidad de Sevilla, Spain,
– Kai T. Salomaa, Queen's University, Kingston, Canada,
– Dragoş Sburlan, Ovidius University of Constanţa, Romania,
– Sergey Verlan, Université Paris XII, France, and
– Claudio Zandron, Università degli Studi di Milano-Bicocca, Italy.

Each submission was reviewed by two independent referees, and from eight papers seven were selected to be presented as regular contributions at the workshop and to be included in this volume. The refereeing work of the members of the Programme Committee is highly appreciated.

In addition to these regular contributions, three invited talks were given by

- Anirban Bandyopadhyay, Advanced Nano Characterisation Center, National Institute for Materials Science, 1-2-1 Sengen, Tsukuba, Ibaraki, Japan 305-0037,
- Giuditta Franco, Università degli Studi di Verona, Italy, and
- Shankara Narayanan Krishna, Department of Computer Science & Engineering, IIT Bombay, Powai, Mumbai, India 400 076.

We are very grateful to the members of the local Organizing Committee, particularly to Aneta Binder, Franziska Gusel, Marion Oswald, and Gabriel Wurzer from the Vienna University of Technology for their invaluable organizational work, and to Gernot Salzer and Sergey Verlan for their support with editing this volume.

The workshop was partially supported by the Institute of Computer Languages of the Vienna University of Technology, the Kurt Gödel Society, the OCG (Austrian Computer Society), and the Austrian Federal Ministry of Science and Research. We extend to all our gratitude.

Vienna, August 2008

Erzsébet Csuhaj-Varjú

Rudolf Freund

Marion Oswald

Kai Salomaa

Table of Contents

Invited Papers

MASSIVE PARALLEL PROCESSING OF PATTERNS ON AN ORGANIC MONOLAYER: TECHNICAL CHALLENGES IN REALISING AN ARTIFICIAL BIO-PROCESSOR 9

Anirban Bandyopadhyay

DNA COMPUTATION: RESULTS, TRENDS, AND PERSPECTIVES 11

Giuditta Franco

ON THE COMPUTATIONAL POWER OF P SYSTEMS WITH WORM OBJECTS . . 15

Shankara Narayanan Krishna

Regular Contributions

MEMBRANE SYSTEMS WITH SURFACE OBJECTS 17

Bogdan Aman and Gabriel Ciobanu

SORTING OMEGA NETWORKS SIMULATED WITH P SYSTEMS: OPTIMAL DATA LAYOUTS 29

Rodica Ceterchi, Mario J. Pérez-Jiménez, and Alexandru Ioan Tomescu

(TISSUE) P SYSTEMS WORKING IN THE k-RESTRICTED MINIMALLY PARALLEL DERIVATION MODE 43

Rudolf Freund and Sergey Verlan

ONE-SIDED INSERTION AND DELETION: TRADITIONAL AND P SYSTEMS CASE 53

Alexander Krassovitskiy, Yurii Rogozhin, and Sergey Verlan

A SMALL UNIVERSAL SPIKING NEURAL P SYSTEM 65

Turlough Neary

AN EXTENDED DOT-BRACKET-NOTATION FOR FUNCTIONAL NUCLEIC ACIDS 75

Effirul I. Ramlan and Klaus-Peter Zauner

A SOFTWARE TOOL FOR GENERATING GRAPHICS BY MEANS OF P SYSTEMS 87

Elena Rivero-Gil, Miguel A. Gutiérrez-Naranjo, Álvaro Romero-Jiménez, Agustín Riscos-Núñez

Author Index 101

MASSIVE PARALLEL PROCESSING OF PATTERNS ON AN ORGANIC MONOLAYER: TECHNICAL CHALLENGES IN REALISING AN ARTIFICIAL BIO-PROCESSOR

Anirban Bandyopadhyay
Advanced Nano Characterisation Center
National Institute for Materials Science
1-2-1 Sengen, Tsukuba, Ibaraki, Japan 305-0037
Email: anirban.bandyo@gmail.com

Neuron-like bio-systems process data in a multilevel way – in terms of $0, 1, 2, 3, \ldots$. To realize a neuron-like multilevel switch $(0, 1, 2, 3, \ldots n)$ in a single molecule we need to control molecular orbital transition between these levels, which has not been realized yet. Once generated, we need to control transport of information through molecules even in atomic scale packing, where wiring is impossible. We have realized such a large flexible connectivity and information processing capabilities in a bi-molecular layer of a quinone derivative on gold (111). The monolayer is found to be extremely flexible towards surviving incredible memory density during transport and logical operation processes. But the existence of natural communication and information exchange among these switching molecules is largely ignored – the enormous possibility hidden in its structure remains unexplored. Several theories like Artificial Neural Networks (ANN), Artificial Intelligence (AI) are developed in the last century but mostly CMOS based giant architectures are used to mimic a few of such basic principles. Prospects of these systems are fairly limited because information processing path is very well defined and connectivity is far less inferior to any biological systems. As an alternative option we have realized tuning molecular properties, and when it survived in ultra-dense packing, then made wireless transport of information to the destination by weak interaction (weak forces of nature serves as connecting wire). As generation and transport occurred at a time, in several parts of the monolayer we created unique memory patterns in a controlled way and observed interaction between these patterns – given the fact that creation of new pattern by interaction is the key to many complex logical and computational features of bio-systems. By pattern manipulation, DDQ monolayer can adapt with situation, organize information by itself, survive under faults – repair without any external help and realize decision making machines (cybernetics, artificial intelligence).

DNA COMPUTATION: RESULTS, TRENDS, AND PERSPECTIVES

Giuditta Franco
Department of Computer Science,
University di Verona, Italy
Email: giuditta.franco@univr.it

An introduction to DNA computing along its most recent trends will open the talk. The issues related to solving NP-complete problems and to DNA encoding yet seem overseen. By DNA self-assembly the three vertex colorability problem has been solved in linear time, and even a programmable transducer was recently built up [1]. Encoding problems were tackled by indirect and more efficient methods, namely in [5], where a set was proposed of non-cross-hybridizing DNA molecules as basic elements, and in [9] where viral sequencing has been used to fold DNA origami of several shapes.

New trends mainly include applications of self-assembly models based on DNA tiles or branched molecules, and construction of DNA nanomachines and walkers. Among the new trends, an interesting one regards the possibility to compute with DNA molecules by means of combinatorial algorithms, having as a side application the introduction of new biotechnological procedures. Indeed, as a matter of fact, several standard biomolecular protocols do not ensure the needed precision for computation. Such techniques have to go through several adjustments, and sometimes completely new protocols are necessary in order to improve their yield. In [6] for example, both a strategy for operon structure optimization by random self-assembly and a DNA comparator to find DNA concentrations were investigated.

An attractive task nowadays is indeed to characterize models within the scope of the given experimental limitations, and to improve or obtain protocols that are sufficiently reliable, controllable and predictable. It is convenient to formulate methods of molecular biology as biomolecular algorithms, and to study the corresponding molecular processes both from combinatoric and experimental points of view. The goal is to optimize the efficiency of biomolecular methodologies, not only to improve the reliability of computing, but also for medical and biological applications ("Optimization methods should be viewed not as vehicles for solving a problem, but for proposing a plausible hypothesis to be confirmed or disconfirmed by further experiments", R. M. Karp, [7]). Namely, an algorithmic analysis of PCR process allowed us to introduce the XPCR technique, which was an effective basis to formulate novel DNA recombination, extraction, and mutagenesis methods [3, 4, 8].

Starting from a heterogeneous pool of DNA double strands sharing a common prefix α and a common suffix β, and given a specified string γ, by means of $XPCR_{\gamma}$, we can recombine all the strings of the pool that contain γ as substring.

The $XPCR_\gamma$ procedure is described by the following four steps.

input a pool P of strings having α as prefix and β as suffix

1. **split** P into P_1 and P_2 (with the same approximate size);
2. **perform** $P_1 := PCR_{(\alpha,\gamma)}(P_1)$ **and** $P_2 := PCR_{(\gamma,\beta)}(P_2)$ (*cutting step*, Figure 1); in order to have the parallelism of these PCRs the encodings of primers must be such that their melting temperatures are approximately the same;

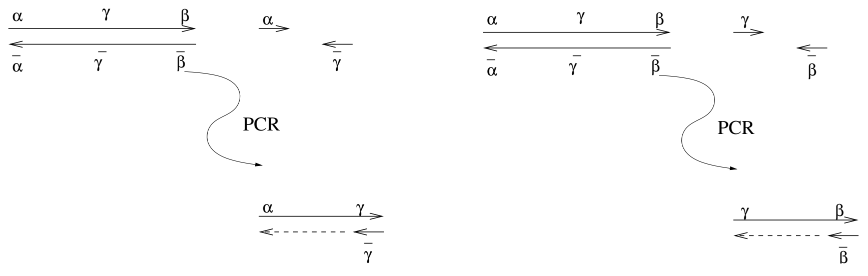


Figure 1: **Cutting step of XPCR$_\gamma$.**

3. **mix** the two pools resulting from the previous step in a new pool $P := P_1 \cup P_2$;
4. **perform** $P := PCR_{(\alpha,\beta)}(P)$ (*recombination step*, see Figure 2);

output the pool P resulting from the previous step.

After the cutting step we find in the test tubes an exponential amplification of the dsDNA $\alpha \dots \gamma$ and $\gamma \dots \beta$, respectively (see Figure 1), that are shorter than the initial molecules (linearly amplified products keep the initial length). In the recombination step, left parts $\alpha \cdots \gamma$ and right parts $\gamma \cdots \beta$ of the sequences of the pool having γ as subsequence are recombined in all possible ways, regardless to the specificity of the sequences between α and γ, or γ and β. Therefore, not only the whole sequences containing γ are restored but also new sequences are generated by recombination (see Figure 2).

From a formal language theoretical viewpoint, XPCR implements null context splicing rules, allowing in principle the actual production of any strictly locally testable language within DNA molecules. Possible algorithmic strategies exploiting XPCR chimeric products will be discussed, namely to generate large libraries, for example for operon structure optimization.

DNA recombination is a very important DNA manipulation, having applications that are beyond the combinatorial mathematical problems. In fact, the production of combinatorial libraries is fundamental to various types of in vitro selection experiments, for selecting new DNA

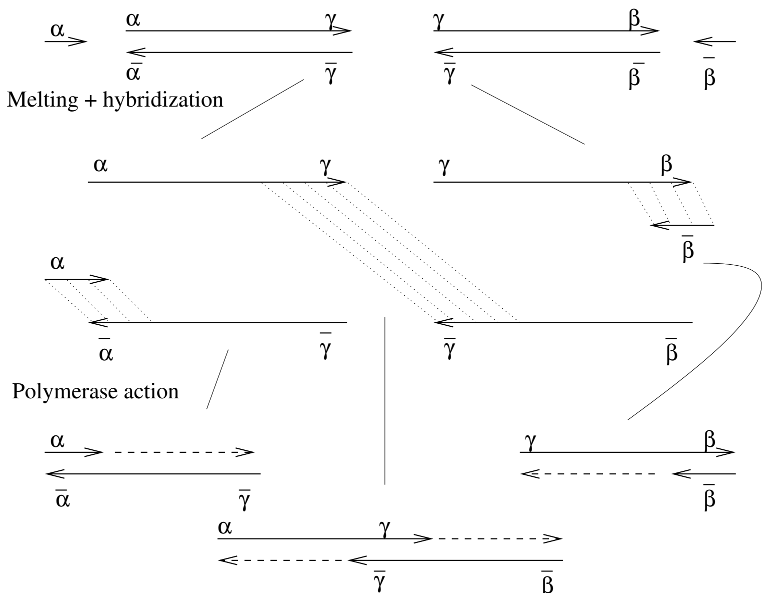


Figure 2: **Recombination basic step of XPCR$_\gamma$.**

or RNA enzymes (such as ribozymes), or for performing crossover of homologous genes and mutagenesis [8]. It is very important also to sequence genomes and to produce DNA memories.

The general schema of a recombination procedure consists of an initial pool with a few different kinds of DNA strands, and a sequence of computational steps that finally produces a DNA pool with a variety of different strands that are made of pieces of the initial strands. A generation method based on XPCR starts from four specific sequences I_1, I_2, I_3, I_4, each constituted by n different strings which can occur in two distinct forms (say X and Y), and, by using only polymerase extension, generates the whole DNA solution space, where all types of possible sequences of the pool are present. This algorithm will be presented along with possible improvements [2], where $\lceil \log_2 n \rceil$ cycles suffices to produce the desired space, that were not experimentally tested yet.

Finally, it will be shown how the study of bioprocesses in combinatorial terms can discover relevant features, as it was the case of recombination witnesses, which are special strands attesting the generation of a whole DNA library by means of a XPCR-based recombination procedure. The existence of the recombination witnesses is a check on the experimental *iter* of our method that represents an actual advantage over all the other existing methods. They provide us with a control of any eventual experimental error which usually cannot be excluded in any DNA procedure.

Some open problems will conclude the talk.

References

[1] CHAKRABORTY, B., JONOSKA, N., SEEMAN, N. C., Programmable transducer by DNA self-assembly, in: [6], 98–99.

[2] FRANCO, G., A polymerase based algorithm for SAT, in: M. Coppo, E. Lodi, G. M. Pinna (Eds.), Theoretical Computer Science, Lecture Notes in Computer Science 3701, 2005, 237–250.

[3] FRANCO, G., GIAGULLI, C., LAUDANNA, C., MANCA, V., DNA extraction by XPCR, in: C. Ferretti, G. Mauri, C. Zandron (Eds.), Proceedings of the 10th International Workshop on DNA Computing: DNA 10, Milan, Italy, June 2004, Revised Selected Papers, Lecture Notes in Computer Science 3384, 2005, 104–112.

[4] FRANCO, G., GIAGULLI, C., LAUDANNA, C., MANCA, V., DNA recombination by XPCR, in: A. Carbone, N.A. Pierce (Eds.), Proceedings of the 11th International Workshop on DNA Computing: DNA 11, London, Ontario, Canada, June 2005, Revised Selected Papers, Lecture Notes in Computer Science 3892, 2006, 55–66.

[5] GARZON, M. H., BOBBA, K. V., HYDE, B. P., Digital information encoding on DNA, in: N. Jonoska, Gh. Păun, G. Rozenberg (Eds.), Aspects of Molecular Computing: Essays Dedicated to Tom Head, on the Occasion of His 70th Birthday, Lecture Notes in Computer Science 2950, 2004, 152–166.

[6] GOEL, A., SIMMEL, F. C., SOSÍK, P. (Eds.), Preliminary Proceedings of the 14th International Meeting on DNA Computing, DNA14, June 2-6, 2008, Prague, Czech Republic.

[7] KARP, R. M., Mapping the genome: some combinatorial problems arising in molecular biology, Proceedings of 25th Annual Symposium on the Theory of Computing, San Diego, CA, USA, May 16-18, New York, NY, USA, ACM, 1993, 278–285.

[8] MANCA, V., FRANCO, G., Computing by polymerase chain reaction, Mathematical Biosciences 211 (2008), 282–298.

[9] ROTHEMUND, P. W. K., Folding DNA to create nanoscale shapes and patterns, Nature 440 (2006), 297–302.

ON THE COMPUTATIONAL POWER OF P SYSTEMS WITH WORM OBJECTS

Shankara Narayanan Krishna
Department of Computer Science & Engineering
IIT Bombay, Powai, Mumbai, India 400 076
Email: `krishnas@cse.iitb.ac.in`

P Systems with worm objects constitute a variant of P systems being universal and able to solve NP-complete problems. This variant processes string objects using the operations of replication, mutation, splitting and recombination. The best result known so far (established by Gh. Păun and C. Martin-Víde) shows that this variant is computationally complete with six membranes. In this work, we study the power of this variant by looking at different combinations of the four operations replication, mutation, splitting and recombination. We show that it is possible to pick some pairs of operations from the four operations mentioned above so that computational completeness can be obtained using only three membranes. We also characterize the power of these systems with only two membranes.

MEMBRANE SYSTEMS WITH SURFACE OBJECTS

Bogdan Aman and Gabriel Ciobanu
"A.I.Cuza" University of Iaşi, Faculty of Computer Science
Blvd. Carol I no.11, 700506 Iaşi, Romania
and
Romanian Academy, Institute of Computer Science
Blvd. Carol I no.8, 700505 Iaşi, Romania
Email: baman@iit.tuiasi.ro, gabriel@info.uaic.ro

Abstract
In this paper we relate the membrane systems with surface objects to a fragment of Brane calculus based on Pino, Phago and Exo operations, by encoding this fragment into the membrane systems with surface objects. We open the discussion on distributing the global multiset of rules of the membrane systems with surface objects into local sets of rules.

1. Introduction

Two recent computational models have been inspired from the structure and the functioning of the living cell: membrane systems [11, 13] and brane calculus [6]. Although the models start from the same observation, they are build having in mind different goals: membrane systems investigate formally the computational nature and power of various features of membranes, while the brane calculus is capable to give a faithful and intuitive representation of the biological reality. In [8] the initiators of these two formalisms describe the goals they had in mind: "While membrane computing is a branch of natural computing which tries to abstract computing models, in the Turing sense, from the structure and the functioning of the cell, making use especially of automata, language, and complexity theoretic tools, brane calculi pay more attention to the fidelity to the biological reality, have as a primary target systems biology, and use especially the framework of process algebra."

A *membrane system* consists of a hierarchy of membranes which do not intersect, with a distinguishable membrane called *skin* surrounding all of them. A membrane without any other membranes inside is *elementary*, while a non-elementary membrane is a *composite* membrane. The membranes define demarcations between *regions*; for each membrane there is a unique associated region. Since we have a one-to-one correspondence, we sometimes use membrane instead of region, and vice-versa. The space outside the skin membrane is called the environment. Regions contain multisets of *objects*, *evolution rules* and possibly other membranes. Only rules in a region delimited by a membrane act on the objects in that region. More details about membrane systems can be found in [13].

Exocytosis is the movement of materials out of a cell via membranous vesicles. These processes allow patches of membrane to flow from compartment to compartment, and require us to think of a cell as a dynamic, rather than static, structure. *Endocytosis* is a general term for a group of processes that bring macromolecules, large particles, small molecules, and even small cells into the eukaryotic cell. There are three types of endocytosis: *pinocytosis*, *phagocytosis* and *receptor-mediated endocytosis*. In all three, the plasma membrane folds inward around materials from the environment, forming a small pocket. The pocket deepens, forming a vesicle. This vesicle separates from the plasma membrane and migrates with its contents to the cell interior.

In brane calculus we have a membrane structure, in which the membranes represent the sites of activity. Opposite to the initial classes of membrane systems in which a computation took place inside the membranes, in brane calculi a computation happens on the membrane. The operations of the two basic brane calculi are directly inspired by biologic processes such as endocytosis, exocytosis and mitosis. The calculus formed using *pino, exo, phago* operations is more expressive then the calculus formed by *mate, drip, bud*, because we can simulate the latter operations using the former ones. Another difference regarding the semantics is expressed in [4]: "whereas brane calculi are usually equipped with an interleaving, sequential semantics (each computational step consists of the execution of a single instruction), the usual semantics in membrane computing is based on maximal parallelism (a computational step is composed of a maximal set of independent interactions)."

Some work was done trying to relate these two models [4, 5, 7, 9]. Inspired by brane calculus, a model of the membrane system having objects attached to the membranes has been introduced in [8]. In [3], a class of membrane systems containing both free floating objects and objects attached to membranes have been proposed, while in [14] a simulation of a bounded symport antiport membrane system using brane calculus is proposed. In this paper we are continuing this research line, and simulate a variant of brane calculus by using membrane systems with surface objects.

The structure of the paper is as follows. In Section 2 we define the class of membrane systems with surface objects used for the simulation, while in Section 3 we present the fragment of brane calculus called PEP. In Section 4 we present an encoding of the PEP calculus into membrane systems with surface objects. The distribution of the global multiset of rules of the membrane systems with surface objects into local sets of rules is discussed in Section 5. Conclusions and references end the paper.

2. Membrane Systems with Surface Objects

The phospholipid bilayer serves as a lipid "lake" in which some proteins "float" (see Figure 1).

Membrane fusion is the process by which a vesicle membrane incorporates its components into the target membrane and releases its cargo into the lumen of the organelle or, in the case of

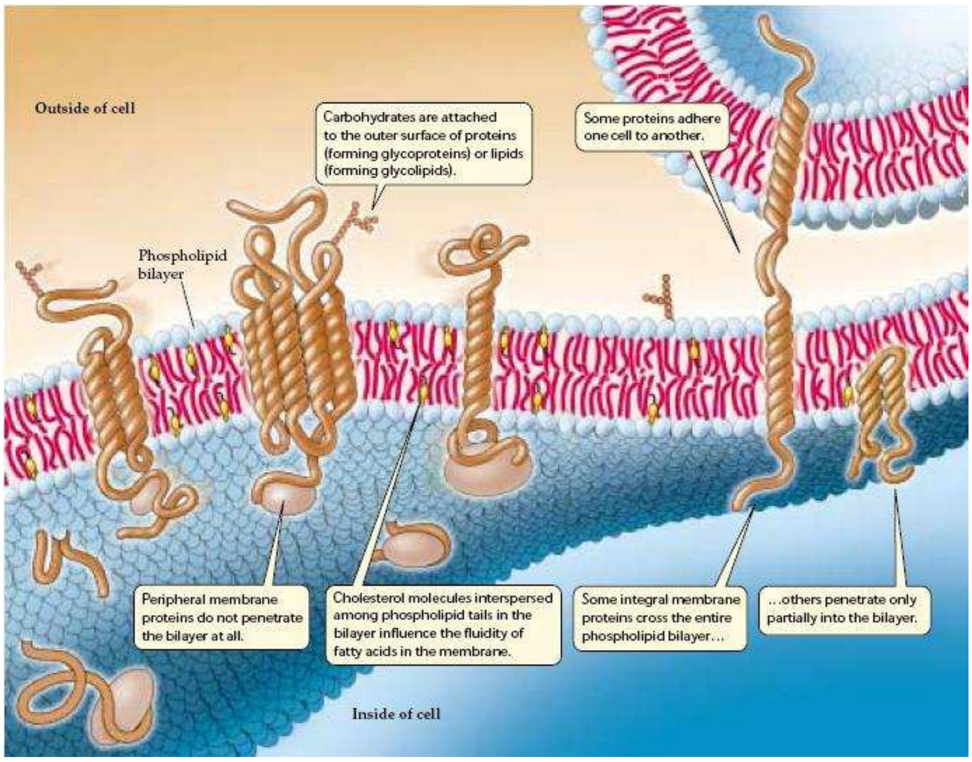

Outside of cell
Carbohydrates are attached to the outer surface of proteins (forming glycoproteins) or lipids (forming glycolipids).
Some proteins adhere one cell to another.
Phospholipid bilayer
Peripheral membrane proteins do not penetrate the bilayer at all.
Cholesterol molecules interspersed among phospholipid tails in the bilayer influence the fluidity of fatty acids in the membrane.
Some integral membrane proteins cross the entire phospholipid bilayer...
...others penetrate only partially into the bilayer.
Inside of cell

Figure 1: **The Fluid Mosaic Model**: The general molecular structure of biological membranes is a continuous phospholipid bilayer in which proteins are embedded.

secretion, into the extracellular medium.

Different steps in membrane fusion are distinguished. First, the vesicle and the target membrane mutually identify each other. Then, proteins from both membranes interact with one another to form stable complexes and bring the two membranes into close apposition, resulting in the docking of the vesicle to the target membrane. Finally, considerable energy needs to be supplied to force the membranes to fuse, since the low-energy organization—in which the hydrophobic tails of the phospholipids are kept away from water while the hydrophilic head groups are in an aqueous medium—must be disrupted, even if only briefly, as the vesicle and target membranes distort and then fuse. Each type of vesicle must only dock with and fuse with the correct target membrane, otherwise the protein constituents of all the different organelles would become mixed with each other and with the plasma membrane.

The molecular processes leading to membrane fusion is only just beginning to take shape; according to [2] two types of proteins, called SNARES and Rab family GTPases work together to achieve the fusion. SNARES located on the vesicles (v-SNARES) and on the target membranes (t-SNARES) interact to form a stable complex that holds the vesicle very close to the target membrane (Fig. 2). Not all vSNARES can interact with all tSNARES, so SNARES provide a first level of specificity. So far, over 50 members of the Rab family have been identified in mammalian cells, and each seems to be found at one particular site where it regulates one

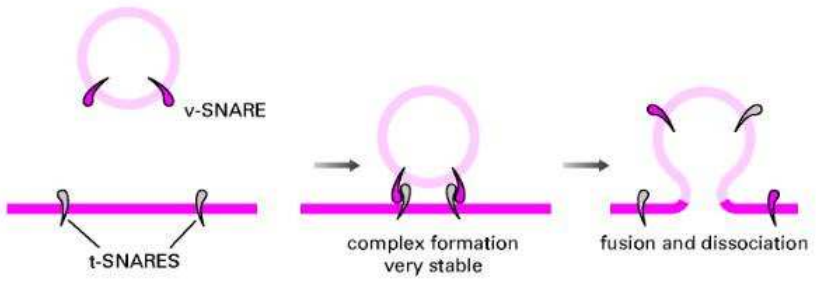


Figure 2: SNAREs and vesicle fusion.

specific transport event, thus controlling which vesicle fuses with which target.

This provides a biological motivation of using objects and co-objects for the exo and phago rules. These rules are also inspired from the approach defined in [8].

We define now the membrane systems with surface objects. Let $\mathbb{N}$ be a set of positive integers, and consider a finite alphabet Γ of symbols. A multiset over Γ is a mapping $u : \Gamma \to \mathbb{N}$. The empty multiset is represented by λ. For any $a \in \Gamma$, the value $u(a)$ denotes the multiplicity of a in u (i.e., the number of occurrences of symbol a in u). Given two multisets u, v over Γ, for any $a \in \Gamma$, we have $(u \uplus v)(a) = u(a) + v(a)$ as the multiset union, and $(u \backslash v)(a) = max\{0, u(a) - v(a)\}$ as the multiset difference. We use the string representation of multisets used in the membrane systems. An example of such a representations $u = aabca$, where $u(a) = 3$, $u(b) = 1$, $u(c) = 1$. Using such a representation, the operations over multisets are defined as operations over strings.

Definition 1. *A membrane system with surface objects (MSO) and n membranes is a construct*

$$\Pi = (A, \mu, u_1, \ldots, u_n, R)$$

where:

1. *A is an alphabet (finite, non-empty) of proteins;*
2. *μ is a membrane structure with $n \geq 2$ membranes;*
3. *$u_1, \ldots, u_n$ are multisets of proteins (represented by strings over A) bound to the n membranes of μ at the beginning of the computation (one assumes that the membranes in μ have a precise identification, e.g., by means of labels, or of other "names", in order to have the marking by means of $u_1, \ldots, u_n$ precisely defined; the labels play no other role than specifying this initial marking of the membranes); the skin membrane is labelled with 1 and $u_1 = \lambda$;*
4. *R is a finite set of rules of the following forms:*

   (a) $[\,]_{vbu} \to_m [[\,]_{vx}]_{uy}$*, where* $b \in A, u, x, y \in A^*, v \in A^+$

pino

The object b creates an empty membrane within the membrane where the b objects is attached and is consumed during the process. The objects v on the empty membrane so created, are transferred from the initial membrane. x and y are newly created multisets of objects.

(b) $[\ ]_{au}[\ ]_{\overline{a}bv} \rightarrow_m [[[\ ]_{ux}]_b]_{vy}$, *where* $a, \overline{a} \in A, u, v, x, y \in A^*$ phago

An object a which also comes with its complementary object $\overline{a}$ models a membrane (the one with $\overline{a}$) "eating" another membrane (the one with a). It proceeds by the membrane containing u wrapping around the membrane containing v and joining itself on the other side. Hence, an additional layer of membrane is created around the eaten membrane: the object on that membrane is b. The objects a and $\overline{a}$ are consumed during the evolution. x and y are newly created multisets of objects.

(c) $[[\ ]_{au}]_{\overline{a}v} \rightarrow_m [\ ]_{uvx}$, *where* $a, \overline{a} \in A, u, v, x \in A^*$ exo

An object a which comes with a complementary object $\overline{a}$ models the merging of two nested membranes, which starts with the membranes touching at a point. In this process (which is a smooth, continuous process), the content of the membrane containing the multiset au gets expelled to the outside, and all objects of the two membranes are united into a multiset on the membrane which initially contained v. The objects a and $\overline{a}$ are consumed during this evolution, and x is a newly created multiset of objects.

2.1. Decision Problems

$A(\Pi)$ denotes the finite alphabet of the system Π, while a marking w represent a distribution of the multiset of objects w over the structure μ of Π. If we consider a multiset of objects w containing all the objects present in the system at a certain moment, then the following proposition holds.

Proposition 1. *It is decidable whether w is a reachable marking of Π, for any MSO system Π and any multiset w of objects over $A(\Pi)$.*

Sketch. Since the alphabet A of Π is finite, we can consider a Petri net having as places the objects from A, as initial marking the multiset $w_0 = u_1 + \ldots + u_n$, and as transitions the way the multiset of objects transform during evolution when applying the *pino*, *exo* and *phago* rules in order to simulate the evolution of the marking of the system Π. Since in a Petri net it is decidable if we can reach a given marking (see Theorem 1), and we have a translation of a system Π into a Petri net, it results that it is decidable if we can reach a given marking w of the system Π. □

Theorem 1 ([10])**.** *For all Petri nets P, for all markings m, m' of P, one can decide whether m' is reachable from m.*

Example 1. *Consider the membrane system with surface object*

$\Pi_1 = (\{a, b, c, \overline{c}\}, [\ ]_1, abc, \{r1 : [\ ]_{abc} \to [[]_{ac}]_{c\overline{c}}, r2 : [[]_{ac}]_{c\overline{c}} \to []_{ac}\})$

The Petri net corresponding to this system is:

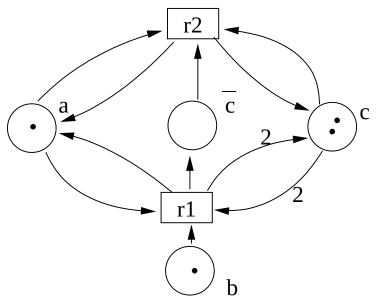


Figure 3: Petri net associated with Π_1

A similar result is presented in [3]; however, the set of rules used is different from the one used in the current paper. The proof of the result presented in [3] is by using grammars without appearance checking, while we use a translation into Petri nets.

2.2. Operational Semantics

Following the line described in [1] where a structural congruence rule was define for membrane systems, we describe here the structural congruence for membrane systems with surface objects.

Table 1: *Syntax of MSO*

Systems	$M, N ::= M\ N \mid [M]_u$	membranes with surface objects
Multisets	$u, v\ ::= \lambda \mid a \mid \overline{a} \mid uv$	multisets of objects where $a, \overline{a} \in A$

We denote by $\mathcal{M}$ the set of systems defined in the above table.
We abbreviate λu as u.

Table 2: *Structural Congruence of MSO*

$M\ N \equiv_m N\ M$	$uv \equiv_m vu$
$M\ (N\ P) \equiv_m (M\ N)\ P$	$u(vw) \equiv_m (uv)w$
	$\lambda u \equiv_m u$
$M \equiv_m N$ implies $M\ P \equiv_m N\ P$	$u \equiv_m v$ implies $uw \equiv_m vw$
$M \equiv_m N$ and $u \equiv_m v$ implies $[M]_u \equiv_m [N]_v$	

The structural congruence relation is a way of rearranging the system such that the interacting parts can come closer. The next rules are added to the rules which appear in Definition 1 in order to show how we can construct the maximal multiset of rules applied in a step of evolution.

Table 3: *Reductions of MSO*

$P \rightarrow_m Q$ implies $P\ R \rightarrow_m Q\ R$	Par
$P \rightarrow_m Q$ implies $[P]_u \rightarrow_m [Q]_u$	Mem
$P \equiv_m P'$ and $P' \rightarrow_m Q'$ and $Q' \equiv_m Q$ implies $P \rightarrow_m Q$	Struct

3. PEP Calculus Without Replication

In this section we give an overview of PEP calculus (phago/exo/pino) without replication; more details can be found in [6]. A membrane structure consists of a collection of nested membranes as can be seen from Table 4. Membranes are formed of patches, where a patch s can be composed from other patches $s = s_1 \mid s_2$. An elementary patch s consists of an action a followed, after the consumption of it, by another patch s_1: $s = a.s_1$. Action often comes in complementary pairs that cause the interaction between membranes. The names n are used to pair-up actions and co-actions. Cardelli motivates that the replication operator is used to model the notion of a "multitude" of components of the same kind, which is in fact a standard situation in biology [6]. We do not use the replicator operator because we are not able to define a membrane system in the following section without knowing exactly the initial membrane structure.

Table 4: *Syntax of Pino/Exo/Phage Calculus*

Systems	$P, Q ::= \diamond \mid P \circ Q \mid \sigma(P)$	nests of membranes
Branes	$\sigma, \tau ::= O \mid \sigma \vert \tau \mid a.\sigma$	combinations of actions
Actions	$a, b ::= n^{\searrow} \mid \overline{n}^{\searrow}(\sigma) \mid n^{\nwarrow} \mid \overline{n}^{\nwarrow} \mid pino(\sigma)$	phago $\searrow$, exo $\nwarrow$

We denote by $\mathcal{P}$ the set of systems defined above. We abbreviate $a.0$ as a, $0(P)$ as (P), and $\sigma(\diamond)$ as $\sigma()$.

Table 5: *Structural Congruence of Pino/Exo/Phage Calculus*

$P \circ Q \equiv_b Q \circ P$	$\sigma \mid \tau \equiv_b \tau \mid \sigma$
$P \circ (Q \circ R) \equiv_b (P \circ Q) \circ R$	$\sigma \mid (\tau \mid \rho) \equiv_b (\sigma \mid \tau) \mid \rho$
$P \circ \diamond \equiv_b P$	$\sigma \mid o \equiv_b \sigma$
$0(\diamond) \equiv_b \diamond$	
$P \equiv_b Q$ implies $P \circ R \equiv_b Q \circ R$	$\sigma \equiv_b \tau$ implies $\sigma \mid \rho \equiv_b \tau \mid \rho$
$P \equiv_b Q$ and $\sigma \equiv_b \tau$ implies $\sigma(P) \equiv_b \tau(Q)$	$\sigma \equiv_b \tau$ implies $a.\sigma \equiv_b a.\tau$

The structural congruence relation is a way of rearranging the system such that the interacting parts come together.

Table 6: *Reductions of Pino/Exo/Phago Calculus*

$pino(\rho).\sigma\vert\sigma_0(P) \rightarrow_b \sigma\vert\sigma_0(\rho(\diamond) \circ P)$	Pino
$\overline{n}^{\nwarrow}.\tau\vert\tau_0(n^{\nwarrow}.\sigma\vert\sigma_0(P) \circ Q) \rightarrow_b P \circ \sigma\vert\sigma_0\vert\tau\vert\tau_0(Q)$	Exo
$n^{\searrow}.\sigma\vert\sigma_0(P) \circ \overline{n}^{\searrow}(\rho).\tau\vert\tau_0(Q) \rightarrow_b \tau\vert\tau_0(\rho(\sigma\vert\sigma_0(P)) \circ Q)$	Phago
$P \rightarrow_b Q$ implies $P \circ R \rightarrow_b Q \circ R$	Par
$P \rightarrow_b Q$ implies $\sigma(P) \rightarrow_b \sigma(Q)$	Mem
$P \equiv_b P'$ and $P' \rightarrow_b Q'$ and $Q' \equiv_b Q$ implies $P \rightarrow_b Q$	Struct

The action $pino(\rho)$ creates an empty bubble within the membrane where the *pino* action resides; we should imagine that the original membrane buckles towards the inside and pinches off. The patch σ on the empty bubble is a parameter of *pino*. The exo action $n^{\nwarrow}$, which comes with a complementary co-action $\overline{n}^{\nwarrow}$, models the merging of two nested membranes, which starts with the membranes touching at a point. In the process (which is a smooth, continuous process), the subsystem P gets expelled to the outside, and all the residual patches of the two membranes become contiguous. The phago action $n^{\searrow}$, which also comes with a complementary co-action $\overline{n}^{\searrow}(\rho)$, models a membrane (the one with Q) "eating" another membrane (the one with P). Again, the process has to be smooth and continuous, so it is biologically implementable. It proceeds by the Q membrane wrapping around the P membrane and joining itself on the other side. Hence, an additional layer of membrane is created around the eaten membrane: the patch on that membrane is specified by the parameter ρ of the cophago action (similar to the parameter of the pino action).

4. Encoding PEP into Membranes with Surface Objects

"At the first sight, the role of objects placed on membranes is different in membrane and brane systems: in membrane computing, the focus is on the evolution of objects themselves, while in brane calculi the objects ("proteins") mainly control the evolution of membranes"[12]. By defining an encoding of the PEP fragment of brane calculus into membranes with surface objects, we show that the difference between the two models is not significant.

Definition 2. *A translation $\mathcal{T} : \mathcal{P} \rightarrow \mathcal{M}$ is given by*

$$\mathcal{T}(P) = \begin{cases} [\mathcal{T}(P)]_{\mathcal{S}(\sigma)} & \textit{if } \sigma(P) \\ \mathcal{T}(Q)\;\mathcal{T}(R) & \textit{if } P = Q \mid R \end{cases}$$

where $\mathcal{S} : \mathcal{P} \rightarrow A$ is defined as:

$$\mathcal{S}(\sigma) = \begin{cases} \sigma & \textit{if } \sigma = n^{\searrow} \textit{ or } \sigma = n^{\nwarrow} \textit{ or } \sigma = \overline{n}^{\nwarrow} \\ \overline{n}^{\searrow}\ S(\rho) & \textit{if } \sigma = \overline{n}^{\searrow}(\rho) \\ \textit{pino}\ S(\rho) & \textit{if } \sigma = \textit{pino}(\rho) \\ \mathcal{S}(a)\ \mathcal{S}(\rho) & \textit{if } \sigma = a.\rho \\ \mathcal{S}(\tau)\ \mathcal{S}(\rho) & \textit{if } \sigma = \tau \mid \rho \end{cases}$$

The rules from the membrane systems with surface objects are of the form:

$[\ ]_{S(n^{\searrow}\sigma|\sigma_0)}[\ ]_{S(\overline{n}^{\searrow}(\rho).\tau|\tau_0)} \rightarrow_m [[[\ ]_{S(\sigma|\sigma_0)}]_{S(\rho)}]_{S(\tau|\tau_0)}$

$[[\ ]_{S(n^{\nwarrow}.\sigma|\sigma_0)}]_{S(\overline{n}^{\nwarrow}.\tau|\tau_0)} \rightarrow_m [\ ]_{S(\sigma|\sigma_0|\tau|\tau_0)}$

$[\ ]_{S(pino(\rho).\sigma|\sigma_0)} \rightarrow_m [[\ ]_{S(\rho)}]_{S(\sigma|\sigma_0)}$

We have the following results:

Proposition 2. *If P is a PEP system and $M = \mathcal{T}(P)$ is a membrane system with surface objects, then there exists N such that $M \equiv_m N$ and $N = \mathcal{T}(Q)$, whenever $P \equiv_b Q$.*

Proposition 3. *If P is a PEP system and $M = \mathcal{T}(P)$ is a membrane system with surface objects, then there exists Q such that $N = \mathcal{T}(Q)$ whenever $M \equiv_m N$.*

Remark 1. *In Proposition 3 it is possible that $P \not\equiv_b Q$. Suppose $P = n^{\searrow}.n^{\nwarrow}(\ )$. By translation we obtain $M = \mathcal{T} = [\ ]_{n^{\searrow}n^{\nwarrow}} \equiv_m [\ ]_{n^{\nwarrow}n^{\searrow}}$=$N$. It is possible to have $Q = n^{\nwarrow}.n^{\searrow}(\ )$ or $Q = n^{\nwarrow}|n^{\searrow}(\ )$ such that $N = \mathcal{T}(Q)$, but $P \not\equiv_b Q$.*

Proposition 4. *If P is a PEP system and $M = \mathcal{T}(P)$ is a membrane system with surface objects, then there exists N such that $M \rightarrow_m N$ and $N = \mathcal{T}(Q)$, whenever $P \rightarrow_b Q$.*

Proposition 5. *If P is a PEP system and $M = \mathcal{T}(P)$ is a membrane system with surface objects, then there exists Q such that $N = \mathcal{T}(Q)$ whenever $M \rightarrow_m N$.*

Remark 2. *In Proposition 5 it is possible that $P \not\rightarrow_b Q$. Suppose $P = \overline{n}^{\nwarrow}.\overline{n}^{\nwarrow}(n^{\nwarrow}.n^{\searrow}(\ ))$. By translation we obtain $M = ((\ )_{n^{\nwarrow}.n^{\searrow}})_{\overline{n}^{\nwarrow}.\overline{n}^{\nwarrow}}$, such that $M \rightarrow_m [\ ]_{n^{\searrow}\overline{n}^{\nwarrow}}$. We observe that there exist $Q = n^{\searrow}.\overline{n}^{\nwarrow}(\ )$ such that $N = \mathcal{T}(Q)$, but $P \not\rightarrow_b Q$.*

The PEP calculus could be extended as in [6] to contain also molecules inside the membranes. Using the new reduction exchanging molecules simultaneous between the interior and exterior of a membrane, the translation can be easily extended by introducing objects in membranes as in [3] and an antiport evolution rule in the definition of $\rightarrow_m$.

5. Global vs. Local Rules

We note in [3, 7, 8] that only a single set of rules is considered in all the definitions of the membranes with surface objects. This means that the rules from this set can be applied anywhere in the system whenever the preconditions are satisfied. This is different from the definition of membrane systems with rewriting rules in which sets of rules are localized in each membrane. Thus if we have a set of rewriting rules and a multiset of objects placed in the same membrane, then the rules may be applied only to this multiset of objects.

Inspired from the class of membrane systems with rewriting rules, we rewrite Definition 1 using different sets of rules placed in each membrane, rules which control only the membranes placed at the first level inside the membranes.

Definition 3. *A localized membrane system with surface objects (LMSO) having* n *membranes is a construct*

$$\Pi = (A, \mu, u_1, \dots, u_n, R_1, \dots, R_n)$$

where:

1. A *is an alphabet (finite, non-empty) of proteins;*
2. μ *is a membrane structure with* $n \geq 2$ *membranes;*
3. $u_1, \dots, u_n$ *are multisets of proteins (represented by strings over* A*) bound to the* n *membranes of* μ *at the beginning of the computation (one assumes that the membranes in* μ *have a precise identification, e.g., by means of labels, or of other "names", in order to have the marking by means of* $u_1, \dots, u_n$ *precisely defined; the labels play no other role than specifying this initial marking of membranes); the skin membrane is labelled with* 1 *and* $u_1 = \lambda$*;*
4. $R_1, \dots, R_n$ *are finite sets of rules of the following forms:*
   (a) $[\ ]_{vbu} \rightarrow_m [[\ ]_{vx}]_{uy}$ pino
   (b) $[[\ ]_{au}]_{\bar{a}v} \rightarrow_m [\ ]_{uvx}$ exo
   (c) $[\ ]_{au}[\ ]_{\bar{a}bv} \rightarrow_m [[[\ ]_{ux}]_b]_{vy}$ phago

In [8] it is mentioned that in case of the *pino* rules, the objects are randomly distributed to the two resulting membranes. In this case what happens to the sets of rules placed in the initial membrane? Are these rules duplicated in both resulting membranes or are distributed randomly?

Definition 4. *Consider a configuration* $\Pi_0 = (A, \mu, u_1, \dots, u_n, R)$ *as in Definition 1. We define the set of reachable configurations by*

$$Reachable(\Pi_0, global) = \{\Pi \mid \Pi_0 \rightarrow_m^* \Pi\}.$$

Definition 5. *Consider a configuration* $\Pi'_0 = (A, \mu, u_1, \dots, u_n, R_1, \dots, R_n)$ *as in Definition 3. We define the set of reachable configurations by*

$$Reachable(\Pi'_0, local) = \{\Pi \mid \Pi'_0 \rightarrow_m^* \Pi\}.$$

If we consider Π_0 and Π'_0 such that they have the same $A, \mu, u_1, \ldots, u_n$, we are interested in investigating what could be the relationship between R and $R_1, \ldots, R_n$ such that the following result holds.

Proposition 6. $\mathit{Reachable}(\Pi_0, \mathit{global}) = \mathit{Reachable}(\Pi'_0, \mathit{local})$

The easiest solution would be to have $R = R_1 = \ldots = R_n$. Each time we create a new membrane by *pino* or *phago*, we create also a duplicate of the set of rules from the membrane which creates the new membrane. Also, when we combine two membranes using *exo*, we merge the two sets of rules in only one. The problem with this solution is that the localized system could become more complicated than the global one.

Depending on how the sets of rules are propagated, we can think that we can obtain other relations between the two multisets (as inclusion, non-empty intersection, etc).

The advantage of the localized systems is that using local rules one would have not to know the entire system during the computation, but only (small) parts of it.

6. Conclusion

We introduce in this paper a new set of rules for membrane systems with surface objects in which we use objects and co-object during the evolution. A novel aspect is given by co-objects. This notion is motivated by the existence of the proteins called SNARES. This aspect is not used in various existing classes of membrane systems using peripheral objects [3, 7, 8]. We show that PEP calculus without replication can be translated into this new class of membrane systems with objects and co-objects. In this way the new class of membranes with surface objects gets the whole computational power of the PEP fragment of the brane calculus.

We have noted that in [3, 7, 8] only a global set of rules is used. We open the discussion on distributing the global multiset of rules into local sets of rules. The equivalence between the global and the localized system remains an open problem.

Acknowledgements

The research of the authors for this paper was partially supported by CEEX 47/2005 and CNCSIS IDEI 402/2007.

References

[1] AMAN, B., CIOBANU, G., Structural properties and observability in membrane systems, SYNASC, IEEE Computer Society (2007), 74-84.

[2] BOLSOVER, S., et. al., Cell Biology. Short Course, Wiley, 2004.

[3] BRIJDER, R., CAVALIERE, M., RISCOS-NÚÑEZ, A., ROZENBERG, G., SBURLAN, D., Membrane systems with marked membranes, Electronic Notes in Theoretical Computer Science 171 (2) (2007), 25–36.

[4] BUSI, N., On the computational power of the mate/bud/drip brane calculus: interleaving vs. maximal parallelism, Workshop on Membrane Computing, Lecture Notes in Computer Science 3850, Springer, 2006, 144–158.

[5] BUSI, N., GORRIERI, R., On the computational power of brane calculi, Third Workshop on Computational Methods in Systems Biology, 2005, 106–117.

[6] CARDELLI, L., Brane calculi. Interactions of biolobical membranes, Lecture Notes in BioInformatics 3082, Springer, 2004, 257–278.

[7] CAVALIERI, M., SEDWARDS, S., Membrane systems with peripherial proteins: transport and evolution, Electronic Notes in Theoretical Computer Science 171(2) (2007), 37–53.

[8] CARDELLI, L., PĂUN, Gh., An universality result for a (mem)brane calculus based on mate/drip operations, ESF Exploratory Workshop on Cellular Computing (Complexity Aspects), Sevilla, 2005, 75–94.

[9] KRISHNA, S. N., Universality results for P systems based on brane calculi operations, Theoretical Computer Science 371 (2007), 83–105.

[10] MAYR, E. W., An algorithm for the general Petri net reachability problem, SIAM Journal of Computing 13(3) (1984), 441-460.

[11] PĂUN, Gh., Computing with membranes, Journal of Computer and System Sciences 61(1) (2000), 108-143.

[12] PĂUN, Gh., Membrane computing and brane calculi (some personal notes), Electronic Notes in Theoretical Computer Science 171 (2007), 3–10.

[13] PĂUN, Gh., Membrane Computing. An Introduction, Springer, 2002.

[14] VITALE, A., MAURI, G., ZANDRON, C., Simulation of a bounded symport antiport P system with brane calculi, Biossytems 91(3) (2008), 558–571.

SORTING OMEGA NETWORKS SIMULATED WITH P SYSTEMS: OPTIMAL DATA LAYOUTS

Rodica Ceterchi[1] Mario J. Pérez-Jiménez[2]
Alexandru Ioan Tomescu[3]

[1]Faculty of Mathematics and Computer Science, University of Bucharest
Academiei 14, RO-010014, Bucharest, Romania
Email: `rceterchi@gmail.com`
[2]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Email: `mario.perez@cs.us.es`
[3]Faculty of Mathematics and Computer Science, University of Bucharest
Academiei 14, RO-010014, Bucharest, Romania
Email: `alexandru.tomescu@gmail.com`

Abstract
The paper introduces some sorting networks and their simulation with P systems, in which each processor/membrane can hold more than one piece of data, and perform operations on them internally. Several data layouts are discussed in this context, and an optimal one is proposed, together with its implementation as a P system with dynamic communication graphs.

1. Introduction

Paper [9] proposed two models to sort a sequence of N numbers, based on the bitonic sorting network. The first one consisted of N membranes, each storing two numbers; one number was an element of the sequence, and the other one was an auxiliary register used to route values. A number x was codified as the number of appearances of a symbol a in each membrane. Moreover, the membranes were disposed on a 2D-mesh, where only communication between neighbor membranes on the mesh are permitted. This model, using a variant of P systems, called P systems with dynamic communication graphs, (see [8]), follows closely the implementation of the bitonic sort on the 2D-mesh.

The second model consisted of only one membrane, where all the N numbers were encoded as occurrences of N different symbols. Restrictions on communication were no longer imposed, as if the underlying communication graph were the complete graph.

In this paper we introduce a model in between the two. First of all, observe that the first model has the advantage of a codifying alphabet of fixed size, while the second has the advantage of a small communication overhead. The model we put forth in this paper captures these two benefits. Each membrane holds a fixed number of values, and each of the membranes can communicate with any other. Additionally, in order to minimize the communication between membranes, we use a periodic remap of values to membranes, according to the steps of the omega network.

The problem of mapping values to processors has been previously addressed in the context of parallel sorting algorithms. The bitonic sorting network, which can sort N keys in time $O(\log^2 N)$, is probably one of the most well-known parallel sorting algorithms. However, modern architectures differ greatly from the theoretical models under which such good results were obtained. As coarse-grained processors can store internally more than one value, the following problem arises: how to map N keys to P processors ($N > P$), such that inter-processor communication is minimized. In the bitonic sorting algorithm, and for $N \geq P^2$, the solution given in [14, 15] consisted in alternating a blocked layout with a cyclic layout, performing thus the minimal number of remaps. This paper gives an optimal mapping strategy for the bitonic sort for any $N > P$, and then applies this result to P systems.

The paper is organized as follows. Section 2 presents preliminaries on bitonic sorting networks and defines omega networks. Section 3 approaches the problem of mapping N keys among P processors, each processor manipulating $n = N/P$ keys, such that overall communication is minimized. Optimal data layouts for the omega network are proposed along the lines of [10], and some essential results are proved about them. Section 4 discusses about internal processing in one processor, and how we model it in our implementation with P systems. Section 5 introduces the P system which simulates the omega network with optimal data layouts, and the algorithms which generate the sequence of dynamic communication graphs of this model. Complexity issues are addressed at the end of Sections 3 and 5.

2. Preliminaries on Bitonic Sorting Networks and Omega Networks

A bitonic sequence is a concatenation of two monotonic sequences, one ascending, and the other one descending, or a sequence such that a cyclic shift of its elements would put them in such a form.

The key components of a bitonic network are the bitonic splitters and the bitonic mergers. The splitter of size N takes as input a bitonic sequence of length N and partitions it in two bitonic sequences of equal length, such that all the elements in the first sequence are smaller than (or greater than) all the elements in the second sequence. A bitonic merger of size N consists of a splitter of size N and of two mergers of size $N/2$, of opposite direction. It accepts as input a bitonic sequence and sorts it in ascending or descending order (direction).

As any sequence of two numbers is bitonic, the sorting network uses bitonic mergers of increasing size and alternating direction to construct bitonic sequences of increasing length. The last such

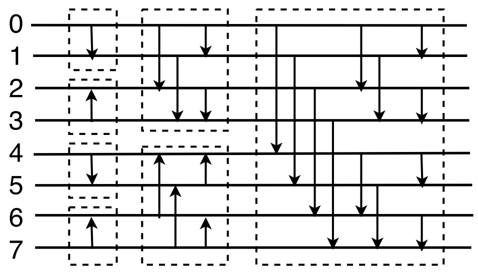
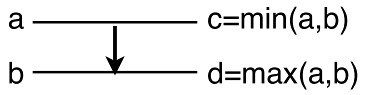
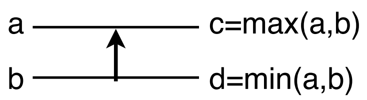


Figure 1: A bitonic sorting network of size $N = 8$. The network can be partitioned in three stages, each containing bitonic mergers of size 2, 4, and 8, respectively.

merger, of size N, renders the whole sequence of N numbers sorted.



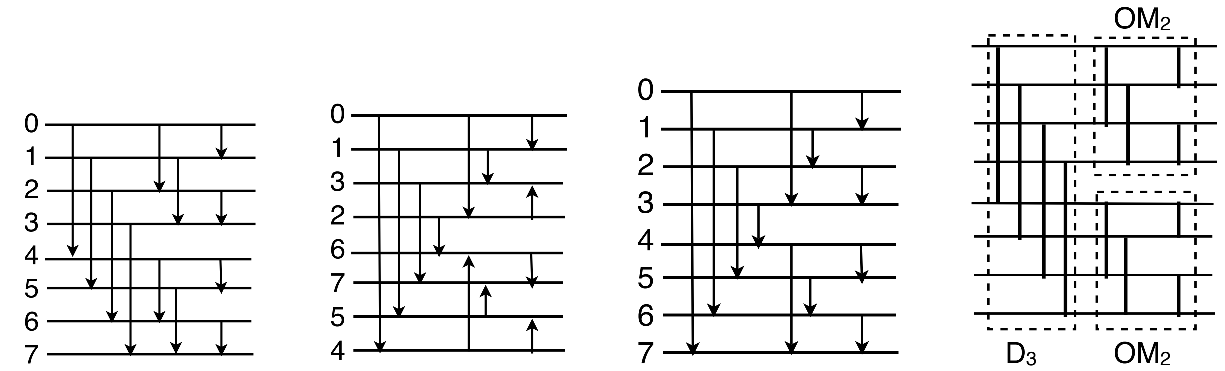
(a) Increasing comparator



(b) Decreasing comparator

Figure 2: Network devices

Following [16] it is customary to represent a network as an ordered set of N *lines* (wires) connected by a set of compare-exchange devices (*comparators*, for brevity). A comparator has two input terminals, a and b, and produces two output terminals c and d. If the comparator is increasing, Fig. 2(a), then $c = \min(a, b)$ and $d = \max(a, b)$, while if the comparator is decreasing, Fig. 2(b), $c = \max(a, b)$ and $d = \min(a, b)$. A bitonic sorting network for $N = 8$ is represented in Fig. 1.



(a) The bitonic merger - classical representation

(b) The bitonic merger - after a permutation of lines

(c) The balanced merger

(d) The omega network OM_3

Figure 3: The bitonic merger, the balanced merger, and the omega network of size 8

We introduce some more notations regarding the serial and parallel connections of networks T_1 and T_2, of size N. Their serial connection, T_1T_2, is a network in which the i-th output terminal of T_1 is connected to the i-th input terminal of T_2. The parallel connection, $T_1 \circ T_2$, is the union of T_1 and T_2, with terminal i of T_1 becoming terminal i of $T_1 \circ T_2$, and terminal i of T_2 becoming terminal $i + N$ of $T_1 \circ T_2$ $(i = 0, \ldots, N-1)$.

Definition 1 (Omega network, Fig. 3(d))**.** *Let D_k, $k \geq 1$ be a one-step network of $N = 2^k$ lines with a device between the pair of lines $(i, i + N/2)$, for $i = 0 \ldots N/2 - 1$. Then the omega network OM_k is recursively defined as $OM_k = D_k(OM_{k-1} \circ OM_{k-1})$.*

In [6] the striking similarity between the bitonic merger (Fig. 3(a) , 3(b)) and the balanced merger (Fig. 3(c)) is investigated. Although prior research [12] showed that there is no permutation of lines to transform the bitonic merger into a balanced merger, a framework is developed under which it is shown that the two mergers are isomorphic graphs, also isomorphic to the graph of the omega network (Fig. 3(d)).

As a serial connection of $\log N$ identical networks in the class of omega networks forms a sorting network [6], in what follows we will concentrate mainly on the omega network.

3. How they communicate

A sorting network is a fine-grained theoretical model, containing exactly one input key on each wire. Additionally, comparators require communication between wires, which can sometimes be more time consuming than the comparison operation itself [2, 3, 11, 17]. When redesigning parallel sorting algorithms for coarse-grained PRAM, one has to pay particular attention to both communication and computation.

Given N keys and P processors $(N > P)$, we have to map $n = N/P$ keys to each processor, such that overall communication is minimized. Ionescu and Schauser [14, 15] investigated this problem for the bitonic sorting algorithm. As initially suggested in [11], they proposed a "smart" periodical switch between a blocked layout and a cyclic layout. They observed that in each stage of the sorting algorithm, the last $\log n$ steps can be performed locally under a blocked layout, while under the cyclic layout the first $\log n$ steps are local. A necessary condition for the two layouts to span enough depth to cover an entire stage of the network is $N \geq P^2$. In addition, the two layouts are particular to the sorting network being implemented. We shall see, for example, that the balanced merger [12, 13], which, as the bitonic merger, belongs to the class of omega networks, also admits data layouts optimizing overall communication.

An approach from the opposite side was put forth by Lee and Batcher [18]. They used a parity strategy for a shared-memory model with $N = 2P$ to store even-parity keys in local memory, while only odd-parity keys were recirculated. This decreased by a factor of 2 the number of shared memory references.

The main contribution of this paper is a general scheme to map N values to P processors, for any $N > P$ and for any sorting network with the topology of the omega network. Our idea captures the essence from the alternating smart layout of [15], and makes it generally applicable, even when $N < P^2$. The number of data layouts is no longer two, but it depends on the granularity of the processors.

3.1. Optimal Data Layouts for the Omega Network

In the following, without explicitly mentioning it, we assume we have to sort $N = 2^k$ keys using P processors, $N > P$, each processor holding $n = N/P$ keys. Any number $i \in \{0, \dots, 2^k-1\}$ has a bit representation $i = a_1 a_2 \cdots a_k$, a_1 being the most significant bit, and a_k the least significant one. To simplify notation, we say that a sequence of bits $a_j \cdots a_i$, where $i, j \in \{1, \dots, k\}$ and $j > i$, stands for the void sequence. The number of parallel steps of OM_k is k, and step t of the omega network OM_k contains devices linking lines whose bit representations differ of bit t, with $1 \leq t \leq k$. For any $t \in \{1, \dots, k\}$, consider the function $bc_t : \{0, 1, \dots, 2^k-1\} \longrightarrow \{0, 1, \dots, 2^k-1\}$, the bit complement of the t-th bit, defined by $bc_t(a_1 a_2 \cdots a_t \cdots a_k) = a_1 a_2 \cdots \bar{a}_t \cdots a_k$. The function bc_t is injective and idempotent.

First, we give a formal definition of a data layout.

Definition 2 (Data layout). *A data layout of N values to P processors is a function $\mathcal{D} : \{0, \dots, N-1\} \to \{0, \dots, P-1\}$.*

We introduce the following data layouts, as suggested in [11, 15].

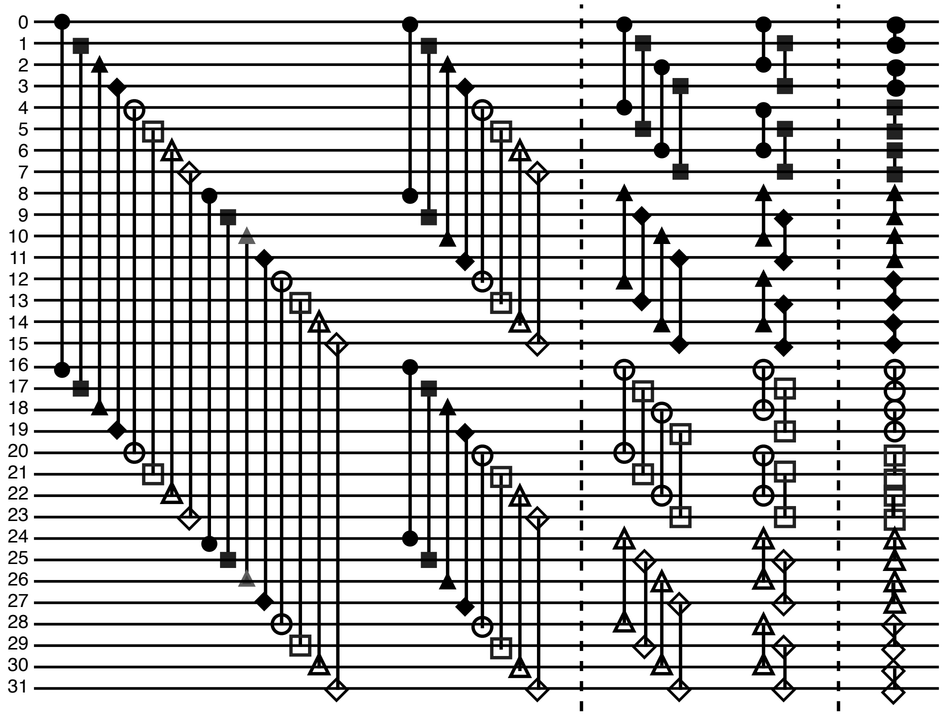
Definition 3 (Blocked layout). *A blocked layout for mapping N keys on P processors is a function $\mathcal{D}^b : \{0, \dots, N-1\} \to \{0, \dots, P-1\}$, such that $\mathcal{D}^b(i) = \lfloor i/n \rfloor$, where $n = N/P$.*

Definition 4 (Cyclic layout). *A cyclic layout for mapping N keys on P processors is a function $\mathcal{D}^c : \{0, \dots, N-1\} \to \{0, \dots, P-1\}$, such that $\mathcal{D}^c(i) = i \bmod P$.*

We note that Definition 5 in [14], and the definition for the cyclic layout indicated in Section 2.1 of [15] are incorrect, since if we map the i-th key to the i mod n processor, where $n = N/P$, we have that $n \leq P$, which implies $N \leq P^2$, which clearly is not the case considered.

In a blocked layout, the first $\log N - \log n$ steps require remote communication, while the last $\log n$ steps are local. In a cyclic layout, the situation is reversed: the first $\log N - \log n$ steps are local, while the last $\log n$ steps are remote. The idea proposed in [15] when mapping $N \geq P^2$ values in the bitonic sort is to periodically switch between the two layouts, such that all steps are local. Moreover, as the stages in a bitonic sort have increasing size, the author proposes an improved "smart" remap such that a layout spans through multiple stages of the algorithm, achieving a total of $\log P + 1$ remaps.

Our paper better highlights the reasoning behind these remaps, in the case of the bitonic sort. Consider the omega network OM_k, and consider we choose to map key 0 to processor 0. If each processor can hold 2^m values, which other keys are mapped to processor 0? As we can see, at step 1 we have a device linking line 0 with line $0 + 2^{k-1}$. At step 2 we have a device linking line



(a) An omega network on size 32. Lines marked with same shape are assigned to the same processor in one data layout.

$\boxed{0\;0}\,0\;0\;0 = 0$
$\boxed{1\;0}\,0\;0\;0 = 16$
$\boxed{0\;1}\,0\;0\;0 = 8$
$\boxed{1\;1}\,0\;0\;0 = 24$

$0\;0\,\boxed{0\;0}\,0 = 0$
$0\;0\,\boxed{1\;0}\,0 = 4$
$0\;0\,\boxed{0\;1}\,0 = 2$
$0\;0\,\boxed{1\;1}\,0 = 6$

$0\;0\;0\,\boxed{0\;0} = 0$
$0\;0\;0\,\boxed{1\;0} = 2$
$0\;0\;0\,\boxed{0\;1} = 1$
$0\;0\;0\,\boxed{1\;1} = 3$

(b) Keys mapped to processor 0 in each of the three data layouts

Figure 4: Three data layouts for the omega network OM_5.

0 with line 2^{k-2}, and a device linking line 2^{k-1} and line $2^{k-1}+2^{k-2}$. We also note that in step 1 lines 2^{k-2} and $2^{k-1}+2^{k-2}$ were also linked with a device. We continue until step m, where we identify 2^m lines linked by 2^{m-1} devices. It would be natural to map these lines to processor 0, as all comparisons at step m are local. However, one more problem remains: all comparisons at stages 0 through $m-1$ are also local? As we shall see, the answer is yes.

The following lemma is straightforward from the definition of OM_k.

Lemma 1. *At each step $1 \le t \le k$ of OM_k, and for any $0 \le i < 2^k$, line i is linked by a device only with line $bc_t(i)$.*

Lemma 2. *In OM_k, for any $0 \le i < 2^{k-m}$, $1 \le m \le k$ and $0 \le t \le k-m$, in steps $t+1,\dots,t+m$ there is no device linking lines in the set $P_i^{t,m} = \{a_1a_2\cdots a_k \mid a_1\cdots a_t a_{t+m+1}\cdots a_k = i$, where $a_1\cdots a_k$ is a bit representation$\}$ with lines from $\{0,\dots,2^k-1\}\setminus P_i^{t,m}$.*

Proof. Suppose there are $1 \le r \le m$, $l \in P_i^{t,m}$ and $l' \notin P_i^{t,m}$ such that at step $t+r$ there is a device linking l and l'. From Lemma 1 we have that $l' = bc_{t+r}(l)$, which implies $l' \in P_i^{t,m}$, a contradiction. □

We can therefore derive the data layouts for the omega network. Suppose we have $N = 2^k$, $n = 2^m$, and $P = 2^{k-m}$. We first assign to each processor P_i all values in the set $P_i^{0,m}$, for $0 \leq i \leq P - 1$. By Lemma 2 we have that the first $\log n = m$ steps are entirely local. After m steps, we remap to each processor P_i all the values in the set $P_i^{m,m}$, and perform the next m stages locally, and so on. We can now give the definition of our proposed data layout.

Definition 5. *Given $N = 2^k$ keys and $P = 2^{k-m}$ processors, which can store $n = 2^m$ values, $m \geq 1$, the sequence of optimal data layouts consists of $\lceil \log N / \log n \rceil = \lceil k/m \rceil$ data layouts. In each data layout $\mathcal{D}_s$, $0 \leq s \leq \lceil k/m \rceil - 1$, values in the set $P_i^{sm,m}$ are mapped to processor P_i, for all $0 \leq i \leq 2^{k-m}$. More formally, for any $0 \leq u < 2^k$ such that $u \in P_i^{sm,m}$, we have $\mathcal{D}_s(u) = i$.*

The following is a consequence of Lemma 1 of [15].

Lemma 3. *The maximum number of successive steps of the omega network that can be executed locally, under any data layout is $\log n$, where $n = N/P$.*

In each data layout $\mathcal{D}_s$, $0 \leq s \leq \lceil k/m \rceil - 2$, $\log n = m$ steps are local. For $s = \lceil k/m \rceil - 1$, the last $k \bmod m$ steps of the network are local. From Lemma 3 we have that the proposed data layouts for the omega network are optimal.

In the case $N \geq P^2$, we notice that $2m > k$, hence two data layouts are enough to cover the whole omega network. However, they do not coincide with $\mathcal{D}^b$ or $\mathcal{D}^c$, as in the blocked layout, the last m stages are local, while in the cyclic layout, the first $k - m$ stages are local.

3.2. Computation Complexity

In each data layout, a processor holds n values and performs $\log n$ steps locally, taking time $O(n \log n)$. As we have $\lceil \log N / \log n \rceil$ data layouts, we get an overall time complexity of the omega network of $O(n \log N)$. From [6] we have that a serial connection of $\log N$ omega networks of size N is enough to sort a sequence of N numbers. Hence, the complexity to sort N numbers using P processors, each holding $n = N/P$ values, using our proposed data layouts, is $O(n \log^2 N)$.

This remark has a quite profound significance. In the fine-grained theoretical model we have $n = 1$, and its complexity is $O(\log^2 N)$. The complexity of the network using a more coarse-grained model depends linearly on the degree of parallelism of the model. At the opposite end, when $n = N$ and the entire sorting network is simulated locally, we have a complexity of $O(N \log^2 N)$, which is worse than $O(N \log N)$, the complexity of most sequential sorting algorithms. It would be desirable to choose n such that this bound is not surpassed in the parallel model. We impose $n \log^2 N \leq N \log N$, which implies $n \leq N / \log N$.

An algorithm to find the minimum of a bitonic sequence of size n in time $O(\log n)$, was introduced in [15]. This gives a time complexity of each data layout of $O(n)$. In the case of a

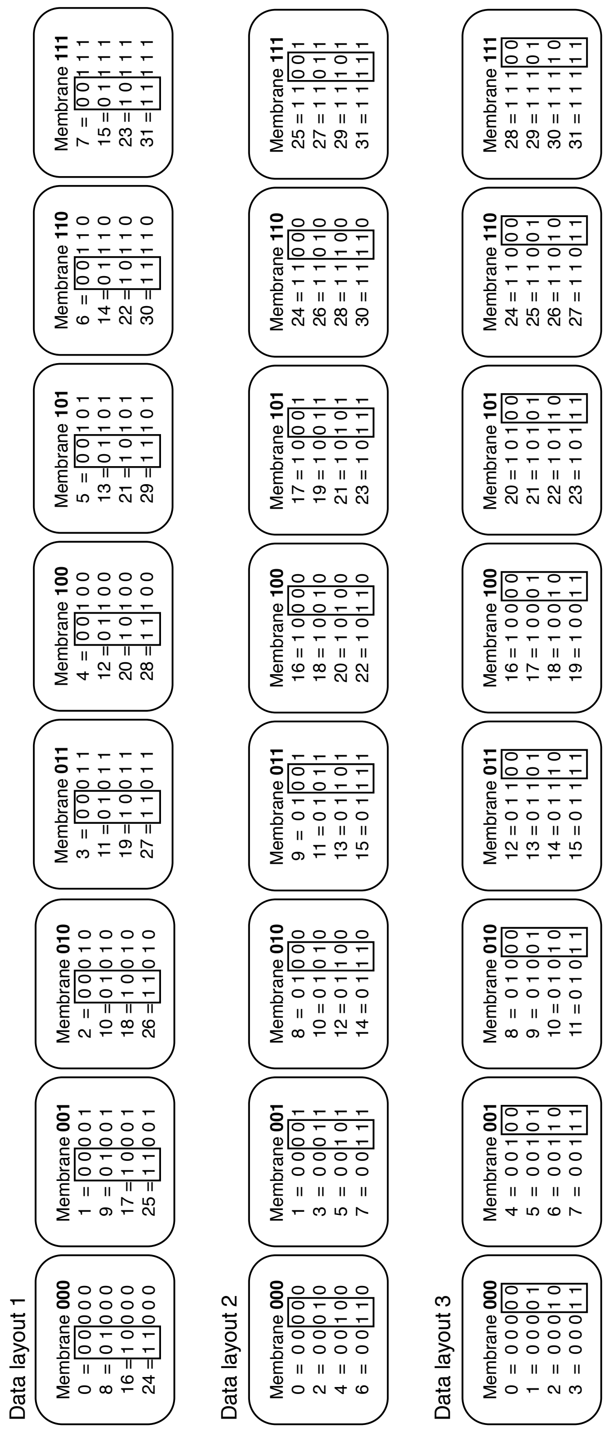


Figure 5: The three data layouts for the omega network in Figure 4(a).

network obtained from a serial connection of bitonic mergers, this observation gives an overall time complexity of $O(\frac{n}{\log n}\log^2 N)$.

4. What happens inside one processor/membrane

One processor (and the membrane which simulates it) will be capable of holding $n = N/P = 2^m$, pieces of data. We label the data with indices in the set $\{0, 1, \cdots, n-1\}$. For any such index we consider its writing as a binary string of length m, for instance $i = x_1 x_2 \cdots x_t \cdots x_m$.

Inside one processor, several comparisons are performed, in parallel, between the n pieces of data, in the following manner: for every bit t, (starting with 1, the most significant bit, and ending with m) we compare and exchange if necessary (to obtain an increasing order) all pairs of values codified with a_i and $a_{bc_t(i)}$. More precisely, we have the following algorithm to be performed inside each processor/membrane:

```
for $t \leftarrow 1$ to $m$ do
    forall $i < bc_t(i)$ in parallel do
        compare($a_i, a_{bc_t(i)}$);
```

Algorithm 1: A parallel algorithm for the bitonic merger

where by **compare**(a_i, a_j) we denote sorting in an ascendant manner the values codified by a_i and a_j, i.e. we end by having the minimum of the two values codified by a_i and the maximum by a_j.

The procedure **compare**(a_i, a_j) works in a membrane in the following manner: let s_i, s_j and t_i, t_j be four auxiliary symbols, for the sources and the targets of a comparator. The set of rules

$$\{a_k \to s_k \mid k = i, j\} \cup \{s_i s_j \to t_i t_j, s_i \to t_j, s_j \to t_j\} \cup \{t_k \to a_k \mid k = i, j\}$$

implement an increasing comparator between values codified by a_i and a_j. We first rewrite the as to ss, next we have the comparator which writes the minimum to t_i and the maximum to t_j, and then we rewrite these back to a_i and a_j respectively.

For all the comparisons which are to be done in parallel, take auxiliary alphabets $S = \{s_0, \cdots, s_{n-1}\}$ and $T = \{t_0, \cdots, t_{n-1}\}$. We rewrite all initial symbols to symbols in S:

$$\{a_i \to s_i \mid i = 0, 1, \cdots, n-1\}.$$

Next we put the comparators between appropriate pairs:

$$\{s_i s_j \to t_i t_j, s_i \to t_j, s_j \to t_j \mid i = 0, 1, \cdots, n-1, i < j = bc_t(i)\}.$$

Then we rewrite back to the original alphabet:

$$\{t_i \to a_i \mid i = 0, 1, \cdots, n-1\}.$$

The parallel comparisons at each step t

```
**forall** $i < bc_t(i)$ **in parallel do**
    **compare**$(a_i, a_{bc_t(i)})$;
```

will thus be simulated in a membrane P by the rules

$$\{a_i \to s_i \mid i = 0, 1, \cdots, n-1\} \cup$$
$$\cup\, \{s_i s_j \to t_i t_j, s_i \to t_j, s_j \to t_j \mid i = 0, 1, \cdots, n-1, i < j = bc_t(i)\} \cup$$
$$\cup\, \{t_i \to a_i \mid i = 0, 1, \cdots, n-1\}.$$

5. A P System which Simulates the Omega Network

In this section we introduce a P system with dynamic communication [7], along the same general lines as the model proposed in [8, 9]. For each of the processors $\mathcal{P}_i$, $i \in \{0, 1, \ldots, P-1\}$ we have an associated membrane, which we label i. The graphs we consider are sub-graphs of the complete graph, K_P, or of the identity graph.

Note that at a certain step of the sorting algorithm not all edges are involved in communication. Therefore we call *active sub-graphs* of K_P those graphs containing only such edges. We also introduce the *identity* graph, with

$$V(Id) = \{0, 1, \ldots, P-1\},$$

$$E(Id) = \{(i, i) \mid 0 \le i \le P-1\}$$

for modeling internal processing steps.

In order to describe the evolution of such a P system, we use pairs of the type $[graph, rules]$. We have $graph$ a sub-graph of K_P or Id and $rules$ a mapping from the set of all edges of $graph$, $E(graph)$, to the set of all symbol/object rewriting rules for routing or comparison operations.

The formal definition of the P system is

$$\Pi =< V = \{a_0, \ldots, a_{n-1}\} \cup \mathcal{A}, \langle [a_0^{x_0^0}, a_1^{x_1^0}, \ldots, a_{n-1}^{x_{n-1}^0}]_0, \ldots, [a_0^{x_0^{P-1}}, a_1^{x_1^{P-1}}, \ldots, a_{n-1}^{x_{n-1}^{P-1}}]_{P-1} \rangle, R_\mu >,$$

where the membrane indices are $\{0, 1, \ldots, P-1\}$. The alphabet $\{a_0, \ldots, a_{n-1}\}$ is of fixed size, and the set $\mathcal{A}$ contains the auxiliary symbols necessary to simulate the omega network, as indicated in Section 4. Numbers x_i^j with $0 \le i \le n-1$ are the values stored on the wires mapped to processor j, $0 \le j \le P-1$ in the first data layout. Each of them is codified as the number of occurrences of a symbol a_i inside membrane j. Finally, R_μ is the finite sequence of pairs $[graph, rules]$ which guides the computation.

We will see in the sequel that R_μ is generated algorithmically, by concatenating sequences of pairs $[graph, rules]$ (we denote the empty sequence by λ, and the concatenation of two sequences by "$\cdot$").

Lemma 4. *Given $N = 2^k$ keys and $P = 2^{k-m}$ membranes, which can store $n = 2^m$ values, $m \geq 1$, after the computation for the data layout $\mathcal{D}_s$ is finished, symbol a_i of membrane j codifies the value corresponding to wire $u \in \{0, \dots, N-1\}$, where the bit representation of u is $u = j_1 \dots j_{sm} i_1 \dots i_m j_{sm+1} \dots j_{k-m}$. By $j_1 \dots j_{k-m}$ and by $i_1 \dots i_m$ we denoted the bit representations of j, and i, respectively.*

Proof. The proof is immediate by Definitions 1, 5 and Lemma 2. □

We observe that the remap of values from a data layout to the other can be done in $P+1$ steps. When passing from data layout $\mathcal{D}_{s-1}$ to $\mathcal{D}_s$, with $0 < s \leq \lceil k/m \rceil - 1$, in each step j, $0 \leq j \leq P-1$, membrane j sends its contents along the edges of the communication graph C_s^j. To avoid collisions in the destination membranes, it also performs a rewriting of symbols from a_t to a'_t, for all $t \in \{0, \dots, n-1\}$. In the last step $P+1$, all auxiliary symbols a'_t will be rewritten back to a_t in all membranes, and the local computation can begin in each membrane.

We give below two algorithms generating the communication graphs C_s^j, and the rules associated to each edge.

```
$E(C_s^j) \leftarrow \emptyset$ ;
for $j \leftarrow 0$ to $P-1$ do
    for $i \leftarrow 0$ to $n-1$ do
        let $j$ have bit representation $j_1 \cdots j_{sm} j_{sm+1} \cdots j_{k-m}$;
        let $i$ have bit representation $i_1 \cdots i_m$;
        // the destination membrane of value encoded by $a_i$ in membrane $j$
        $z \leftarrow j_1 \cdots j_{sm} i_1 \cdots i_m j_{(s+1)m+1} \cdots j_{k-m}$;
        // the destination symbol of value encoded by $a_i$ in membrane $j$
        $t \leftarrow j_{sm+1} \cdots j_{sm+m}$;
        $E(C_s^j) := E(C_s^j) \cup \{j, z\}$;
        $\text{rules}_{C_s^j}((j, z)) := a_i \rightarrow a'_t$ ;
```

Algorithm 2: Generation of the sequence of P communication graphs when passing from data layout $\mathcal{D}_{s-1}$ to $\mathcal{D}_s$, with $0 < s \leq \lceil k/m \rceil - 1$.

```
for $j \leftarrow 0$ to $P-1$ do
    rules-endcomm$((j,j)) := \{a'_i \rightarrow a_i \mid 0 \leq i \leq n-1\}$;
```

Algorithm 3: Generation of the rules associated to the identity graph which rewrite back the auxiliary symbols a'_t when passing from any data layout $\mathcal{D}_{s-1}$ to $\mathcal{D}_s$, with $0 < s \leq \lceil k/m \rceil - 1$.

We assume that the sequence denoted by *SimOM* is the sequence of pairs $[graph, rules]$ which simulates the omega network of size n, OM_m ($n = 2^m$). Its construction was indicated in Section 4 and is expressed algorithmically below.

```
$SimOM \leftarrow \lambda$;
for $t \leftarrow 1$ to $m = \log n$ do
    forall $p \leftarrow 0$ to $P-1$ in parallel do
        rules$_{t,1}((p,p)) \leftarrow \{a_i \rightarrow s_i \mid i = 0, 1, \ldots n-1\}$;
        rules$_{t,2}((p,p)) \leftarrow \{s_i s_j \rightarrow t_i t_j, s_i \rightarrow t_j, s_j \rightarrow t_j \mid i = 0, 1, \ldots, n-1, i < j = bc_t(i)\}$;
        rules$_{t,3}((p,p)) \leftarrow \{t_i \rightarrow a_i \mid i = 0, 1, \cdots n-1\}$;
    $SimOM \leftarrow SimOM \cdot [Id$,rules$_{t,1}] \cdot [Id$,rules$_{t,2}] \cdot [Id$,rules$_{t,3}]$;
```

Algorithm 4: Generation of the sequence *SimOM* which simulates the omega network of size n.

We can now give the algorithm which generates the whole sequence R_μ guiding the computation.

```
$R_\mu \leftarrow \lambda$;
for $s \leftarrow 1$ to $\lceil k/m \rceil - 1$ do
    $R_\mu \leftarrow R_\mu \cdot SimOM$;
    for $j \leftarrow 0$ to $P-1$ do
        $R_\mu \leftarrow R_\mu \cdot [C_s^j$, rules$_{C_s^j}]$;
    $R_\mu \leftarrow R_\mu \cdot [Id$, rules-endcomm$]$;
$R_\mu \leftarrow R_\mu \cdot SimOM$;
```

Algorithm 5: Generation of the sequence R_μ which guides the computation.

5.1. Computation complexity

Observe that the length of the sequence *SimOM* is $3 \log n$. As we have $\frac{\log N}{\log n}$ data layouts, and that in each data layout $3 \log n$ steps are needed for *SimOM* and another $P+1$ steps are needed for communication, the length of R_μ is $3 \log N + \frac{N \log N}{n \log n}$. A sorting network can be obtained by a serial connection of $\log N$ omega networks, hence our model can sort in time $O(\log^2 N + \frac{N \log^2 N}{n \log n})$. Note that when $n = N$ all computation is local, and the complexity is the best possible, $O(\log^2 N)$. When $n = 2$ the complexity increases to $O(N \log^2 N)$.

Acknowledgements

We thank the anonymous referees for their comments and suggestions which helped improve the quality of the paper.

References

[1] AJTAI, M., KOMLOS, J., SZEMEREDI, E., An $O(N \log N)$ sorting network, Proc. 15th Ann. ACM Symp. Theory of Computing, 1983, 1–9.

[2] AGGARWAL, A., CHANDRA, A. K., SNIR, M., Communication complexity of PRAMs, Theoretical Computer Science 71 (1) (1990), 3–28.

[3] ALEXANDROV, A., IONESCU, M., SCHAUSER, K. E., SCHEIMAN, C., LogGP: incorporating long messages into the LogP model, Journal of parallel and distributed computing 44 (1) (1997), 71–79.

[4] ALHAZOV, A., SBURLAN, D., Static sorting P systems, Ch. 8 in: G. Ciobanu, Gh. Păun, M. J. Pérez Jiménez (Eds.), Applications of Membrane Computing, Springer, 2005.

[5] BATCHER, K. E., Sorting networks and their applications, Proc. AFIPS Spring Joint Comput. Conf. 32 (1968), 307–314.

[6] BILARDI, G., Merging and sorting networks with the topology of the Omega Network, IEEE Transactions on Computers 38 (10) (1989), 1396–1403.

[7] CETERCHI, R., MARTÍN-VIDE, C., Dynamic P systems, Lecture Notes in Computer Science 2597, 2003, 146–186.

[8] CETERCHI, R., PÉREZ JIMÉNEZ, M. J., On two-dimensional mesh networks and their simulation with P systems, Lecture Notes in Computer Science 3365, 2005, 259–277.

[9] CETERCHI, R., PÉREZ JIMÉNEZ, M. J., TOMESCU, A. I., Simulating the Bitonic Sort Using P Systems, in: G. Eleftherakis et al. (Eds.), WMC8 2007, Lecture Notes in Computer Science 4860, 2007, 172–192.

[10] CETERCHI, R., TOMESCU, A. I., Implementing sorting networks with spiking neural P systems, Fundamenta Informaticae, 2008, to appear.

[11] CULLER, D. E., KARP, R. M., PATTERSON, D. A., SAHAY, A., SCHAUSER, K. E., SANTOS, E., SUBRAMONIAN, R., VON EICKEN, T., LogP: Towards a realistic model of parallel computation, Proc. Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 1993, 1–12.

[12] DOWD, M., PERL, Y., SAKS, M., RUDOLPH, L., The balanced sorting network, Proc. Second annual ACM symp. on Principles of distributed computing, 1983, 161–172.

[13] DOWD, M., PERL, Y., SAKS, M., RUDOLPH, L., The periodic balanced sorting network, JACM 36. (4) (1989), 738–757.

[14] IONESCU, M. F., Optimizing parallel bitonic sort, Tech. Report TRCS96-14, Dept. of Comp. Sci., Univ. of California, Santa Barbara, 1996.

[15] IONESCU, M. F., SCHAUSER, K. E., Optimizing parallel bitonic sort, Proc. 11th Int'l Parallel Processing Symp., 1997, 303–309.

[16] KNUTH, D. E., The Art of Computer Programming, volume 3: Sorting and Searching, second ed., Redwood City, CA, Addison Wesley Longman, 1998.

[17] KRUSKAL, C., RUDOLPH, L., SNIR, M., A complexity theory of efficient parallel algorithms, Theoretical Computer Science 71 (1) (1990), 95–132.

[18] LEE, J. D., BATCHER, K. E., Minimizing communication in the bitonic sort, IEEE Trans. on Parallel and Distributed Systems 11 (5) (2000), 459–474.

[19] LEIGHTON, F., Tight bounds on the complexity of parallel sorting, IEEE Trans. Computers 34 (4) (1985), 344–354.

[20] PATERSON, M. S., Improved sorting networks with $O(\log N)$ depth, Algorithmica 5 (1990), 75–92.

(TISSUE) P SYSTEMS WORKING IN THE k-RESTRICTED MINIMALLY PARALLEL DERIVATION MODE

Rudolf Freund[1] and Sergey Verlan[2]

[1]Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
Email: `rudi@emcc.at`
[2]LACL, Département Informatique
UFR Sciences et Technologie, Université Paris XII
61, av. Général de Gaulle, 94010 Créteil, France
Email: `verlan@univ-paris12.fr`

Abstract
Just recently several new derivaton modes for (tissue) P systems have been introduced and investigated in addition to the maximally parallel derivation mode used from the beginning in the area of membrane computing. A variant of the minimally parallel derivation is considered in this paper – we allow only a bounded number of rules to be taken from every set of the partitioning of the whole set of rules. The 1*-restricted minimally parallel derivation mode especially fits to describe the way derivations take place in spiking neural P systems without delays, i.e., in every neuron where a rule is applicable exactly one rule has to be applied. Moreover, purely catalytic P systems working in the maximally parallel derivation mode can be described as P systems using the corresponding rules without catalysts when working in the* 1*-restricted minimally parallel derivation mode.*

1. Introduction

In the original model of P systems introduced as membrane systems by Gh. Păun (see [5], [10]), the objects evolve in a hierarchical membrane structure; in tissue P systems, first considered by Gh. Păun and T. Yokomori in [13] and [14], also see [7], the cells communicate within an arbitrary graph topology. In the original model of membrane systems as well as in many variants of P systems and tissue P systems investigated during the last decade, the maximally parallel derivation mode was used. Just recently several new derivation modes for P systems and tissue P systems have been introduced and investigated, for example, the sequential and the asynchronous derivation mode as well as the minimally parallel derivation mode (see [3]). In [8], a formal framework for (tissue) P systems capturing the formal features of these derivation modes was developed, based on a general model of membrane systems as a collection of interacting cells containing multisets of objects (compare with the models of networks of cells as discussed in [2] and networks of language processors as considered in [4]). Continuing the formal approach started in [8], a variant of the minimally parallel derivation mode is considered in this paper: In the minimally parallel derivation mode, we consider a partitioning of the whole set of rules and allow only multisets of rules to be applied in parallel which cannot be extended by adding a rule from a set of rules from which no rule has already been taken into this multiset of rules. Whereas in the minimally parallel derivation mode, an arbitrary number of rules can be used from any of the sets of rules in the partitioning of rules, only a bounded number of at most k rules can be taken from each of these sets of the partitioning in the k-restricted minimally parallel derivation mode.

The rest of this paper now is organized as follows: In the second section, well-known definitions and notions are recalled. In the next section, we consider a general class of multiset rewriting systems containing, in particular, many variants of P systems and tissue P systems as well as even (extended) spiking neural P systems without delays and give formal definitions of the most important well-known derivation modes (maximally parallel, minimally parallel) as well as the new k-restricted minimally parallel derivation mode. The 1-restricted minimally parallel derivation mode especially fits to describe the way derivations take place in spiking neural P systems without delays, i.e., in every neuron where a rule is applicable exactly one rule has to be applied. Moreover, purely catalytic P systems working in the maximally parallel derivation mode can be described as P systems using the corresponding rules without catalysts when working in the 1-restricted minimally parallel derivation mode. An outlook to future research topics involving the k-restricted minimally parallel derivation mode concludes the paper.

2. Preliminaries

We recall some of the notions and the notations we use (see [12] for elements of formal language theory). Let V be a (finite) alphabet; then V^* is the set of all strings (a language) over V, and $V^+ = V^* - \{\lambda\}$ where λ denotes the empty string. RE, REG ($RE\,(T)$, $REG\,(T)$) denote the families of recursively enumerable and regular languages (over the alphabet T), respectively. For any family of string languages F, PsF denotes the family of Parikh sets of languages from F and NF the family of Parikh sets of languages from F over a one-letter alphabet. By $\mathbb{N}$ we denote the set of all non-negative integers, by $\mathbb{N}^k$ the set of all vectors of non-negative integers; $[k..m]$ for $k \leq m$ denotes the set of natural numbers n with $k \leq n \leq m$. In the following, we will not distinguish between NRE, which coincides with $PsRE\,(\{a\})$, and $RE\,(\{a\})$.

Let V be a (finite) set, $V = \{a_1, ..., a_k\}$. A *finite multiset* M over V is a mapping $M : V \longrightarrow \mathbb{N}$, i.e., for each $a \in V$, $M\,(a)$ specifies the number of occurrences of a in M. The size of the multiset M is $|M| = \sum_{a \in V} M\,(a)$. A multiset M over V can also be represented by any string x that contains exactly $M\,(a_i)$ symbols a_i for all $1 \leq i \leq k$, e.g., by $a_1^{M(a_1)}...a_k^{M(a_k)}$, or else by the set $\left\{a_i^{M(a_i)} \mid 1 \leq i \leq k\right\}$. The support of M is the set $supp(M) = \{a \in V \mid f(a) \geq 1\}$.

The set of all finite multisets over the set V is denoted by $\langle V, \mathbb{N}\rangle$. We may also consider mappings M of the form $M : V \longrightarrow \mathbb{N}_\infty$ where $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$, i.e., elements of M may have an infinite multiplicity; we shall call such multisets where $M(a_i) = \infty$ for at least one i, $1 \leq i \leq k$, *infinite multisets.* The set of all such multisets M over V with $M : V \longrightarrow \mathbb{N}_\infty$ is denoted by $\langle V, \mathbb{N}_\infty \rangle$. For $W \subseteq V$, W^∞ denotes the infinite multiset with $W(a) = \infty$ for all $a \in W$.

Let x and y be two multisets over V, i.e., from $\langle V, \mathbb{N} \rangle$ or $\langle V, \mathbb{N}_\infty \rangle$. Then x is called a submultiset of y, written $x \leq y$ or $x \subseteq y$, if and only if $x(a) \leq y(a)$ for all $a \in V$; if, moreover, $x(a) < y(a)$ for some $a \in V$, then x is called a strict submultiset of y. Observe that for all $n \in \mathbb{N}$, $n+\infty = \infty$, and $\infty - n = \infty$. The sum of x and y, denoted by $x + y$ or $x \cup y$, is a multiset z such that $z(a) = x(a) + y(a)$ for all $a \in V$. The difference of two multisets x and y, denoted by $x - y$, provided that $y \subseteq x$, is the multiset z with $z(a) = x(a) - y(a)$ for all $a \in V$. Observe that in the following, when taking the sum or the difference of two multisets x and y from $\langle V, \mathbb{N}_\infty \rangle$, we shall always assume $\{x(a), y(a)\} \cap \mathbb{N} \neq \emptyset$.

If $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_m)$ are vectors of multisets over V, then $X \leq Y$ if and only if $x_j \subseteq y_j$ for all j, $1 \leq j \leq m$; in the same way, sum and difference of vectors of multisets are defined by taking the sum and the difference, respectively, in each component.

Throughout the rest of the paper, we will not distinguish between a multiset from $\langle V, \mathbb{N} \rangle$ and its representation by a string over V containing the corresponding number of each symbol. Moreover, when we speak of a partitioning of a set R into a set $\{R_i \mid 1 \leq i \leq h\}$ of subsets of R, i.e., $R_i \subseteq R$, $1 \leq i \leq h$, the R_i are not necessarily disjoint.

3. Networks of Cells

In this section we consider membrane systems as a collection of interacting cells containing multisets of objects like in [2] and [8]. For an introduction to the area of membrane computing we refer the interested reader to the monograph [11], the actual state of the art can be seen in the web [15].

Definition 3.1. A *network of cells with checking sets of degree* $n \geq 1$ is a construct

$$\Pi = (n, V, w, R)$$

where

1. n is the number of cells;

2. V a *finite alphabet*;

3. $w = (w_1, \ldots, w_n)$ where $w_i \in \langle V, \mathbb{N}_\infty \rangle$, for all $1 \leq i \leq n$, is the *multiset initially associated to cell* (in most of the cases, at most one cell, then being called *the environment*, will contain symbols occurring with infinite multiplicity);

4. R is a finite set of *interaction rules* of the form

$$(E : X \to Y)$$

where E is a recursive condition for configurations of Π (see definition below) as well as $X = (x_1, \dots, x_n)$, $Y = (y_1, \dots, y_n)$, with $x_i, y_i \in \langle V, \mathbb{N} \rangle$, $1 \leq i \leq n$, are vectors of multisets over V. We will also use the notation

$$(E : (x_1, 1) \dots (x_n, n) \to (y_1, 1) \dots (y_n, n))$$

for a rule $(E : X \to Y)$.

A network of cells consists of n cells, numbered from 1 to n, that contain (possibly infinite) multisets of objects over V; initially cell i contains w_i. A *configuration* C of Π is an n-tuple of multisets over V $(u_1, \dots, u_n)$; the *initial configuration* of Π, C_0, is described by w, i.e., $C_0 = w = (w_1, \dots, w_n)$. Cells can interact with each other by means of the rules in R. An interaction rule

$$(E : (x_1, 1) \dots (x_n, n) \to (y_1, 1) \dots (y_n, n))$$

is applicable to a configuration C if and only if C fulfills condition E; its application means rewriting objects x_i from cells i into objects y_j in cells j, $1 \leq i, j \leq n$.

The set of all multisets of rules *applicable* to C is denoted by $Appl(\Pi, C)$ (a procedural algorithm how to obtain $Appl(\Pi, C)$ is described in [8]).

For the specific *derivation modes* to be defined in the following, the selection of multisets of rules applicable to a configuration C has to be a specific subset of $Appl(\Pi, C)$; for the derivation mode ϑ, the selection of multisets of rules applicable to a configuration C is denoted by $Appl(\Pi, C, \vartheta)$.

Definition 3.2. For the *asynchronous* derivation mode (*asyn*),

$$Appl(\Pi, C, asyn) = Appl(\Pi, C),$$

i.e., there are no particular restrictions on the multisets of rules applicable to C.

Definition 3.3. For the *sequential* derivation mode (*sequ*),

$$Appl(\Pi, C, sequ) = \{R' \mid R' \in Appl(\Pi, C) \text{ and } |R'| = 1\},$$

i.e., any multiset of rules $R' \in Appl(\Pi, C, sequ)$ has size 1.

The most important derivation mode considered in the area of P systems from the beginning is the *maximally parallel* derivation mode where we only select multisets of rules R' that are not extensible, i.e., there is no other multiset of rules $R'' \supsetneqq R'$ applicable to C.

Definition 3.4. For the *maximally parallel* derivation mode (max),

$$\begin{aligned} Appl\,(\Pi, C, max) \;=\; & \{R' \mid R' \in Appl\,(\Pi, C) \text{ and there is} \\ & \text{no } R'' \in Appl\,(\Pi, C) \text{ with } R'' \supsetneqq R'\}\,. \end{aligned}$$

For the *minimally parallel* derivation mode, we need an additional feature for the set of rules R, i.e., we consider a partition of R into disjoint subsets R_1 to R_h. Usually, this partition of R may coincide with a specific assignment of the rules to the cells. For any set of rules $R' \subseteq R$, let $\|R'\|$ denote the number of sets of rules R_j, $1 \leq j \leq h$, with $R_j \cap R' \neq \emptyset$.

There are several possible interpretations of this minimally parallel derivation mode which in an informal way can be described as applying multisets such that from every set R_j, $1 \leq j \leq h$, at least one rule – if possible – has to be used (e.g., see [3]). For the basic variant as defined in the following, in each derivation step we choose a multiset of rules R' from $Appl\,(\Pi, C, asyn)$ that cannot be extended to $R'' \in Appl\,(\Pi, C, asyn)$ with $R'' \supsetneqq R'$ as well as $(R'' - R') \cap R_j \neq \emptyset$ and $R' \cap R_j = \emptyset$ for some j, $1 \leq j \leq h$, i.e., extended by a rule from a set of rules R_j from which no rule has been taken into R'.

Definition 3.5. For the *minimally parallel* derivation mode (min),

$$\begin{aligned} Appl\,(\Pi, C, min) \;=\; & \{R' \mid R' \in Appl\,(\Pi, C, asyn) \text{ and} \\ & \text{there is no } R'' \in Appl\,(\Pi, C, asyn) \\ & \text{with } R'' \supsetneqq R',\ (R'' - R') \cap R_j \neq \emptyset \\ & \text{and } R' \cap R_j = \emptyset \text{ for some } j,\ 1 \leq j \leq h\}\,. \end{aligned}$$

In [8], further restricting conditions on the four basic modes defined above, especially interesting for the minimally parallel derivation mode, were considered. We now consider a restricted variant of the minimally parallel derivation mode allowing only a bounded number of at most k rules to be taken from each set R_j, $1 \leq j \leq h$, of the partitioning into a multiset of rules applicable in the minimally parallel derivation mode.

Definition 3.6. For the *k-restricted minimally parallel* derivation mode (min_k),

$$\begin{aligned} Appl\,(\Pi, C, min_k) \;=\; & \{R' \mid R' \in Appl\,(\Pi, C, min) \text{ and} \\ & |R' \cap R_j| \leq k \text{ for all } j,\ 1 \leq j \leq h\}\,. \end{aligned}$$

For all the derivation modes defined above, we now can define how to obtain a next configuration from a given one by applying an applicable multiset of rules according to the constraints of the underlying derivation mode:

Definition 3.7. Given a configuration C of Π and a derivation mode ϑ, we may choose a multiset of rules $R' \in Appl\,(\Pi, C, \vartheta)$ in a non-deterministic way and apply it to C. The result of this *transition step* from the configuration C with applying R' is the configuration $Apply\,(\Pi, C, R')$, and we also write $C \Longrightarrow_{(\Pi,\vartheta)} C'$. The reflexive and transitive closure of the transition relation $\Longrightarrow_{(\Pi,\vartheta)}$ is denoted by $\Longrightarrow^*_{(\Pi,\vartheta)}$.

Definition 3.8. A *computation* in a network of cells Π, $\Pi = (n, V, w, R)$, starts with the initial configuration $C_0 = w$ and continues with transition steps according to the chosen derivation mode ϑ; it is called *successful* if we reach a configuration C to which no multiset of rules can be applied with respect to the derivation mode ϑ anymore, i.e., $Appl\,(\Pi, C, \vartheta) = \emptyset$ (we also say that the computation *halts*).

As the results of a halting computation we take the number of objects in a specified output cell. We shall use the notation

$$O_m C_n\,(\vartheta)\,[\text{parameters for rules}]$$

to denote the family of sets of natural numbers generated by networks of cells $\Pi = (n, V, w, R)$ with $m = |V|$ as well as ϑ indicating the derivation mode; the *parameters for rules* describe the specific features of the rules in R. If any of the parameters m and n is unbounded, we replace it by $*$.

4. Specific Examples for the 1-Restricted Minimally Parallel Derivation Mode

In this section, we show how the 1-restricted minimally parallel derivation mode may capture characteristic features of well-known models of P systems.

We first consider extended spiking neural P systems (without delays, see [1]), where the rules are applied in a sequential way in each neuron, but on the level of the whole system, the maximally parallel derivation mode is applied (every neuron which may use a spiking rule has to spike, i.e., to apply a rule, see the original paper [9]). When partitioning the rule set according to the set of neurons, the application of the 1-restricted minimally parallel derivation mode exactly models the original derivation mode defined for spiking neural P systems.

Already in the original paper of Gh. Păun (see [10]), membrane systems with catalytic rules were defined, but used together with other noncooperative rules. In [6] it was shown that only three catalysts are sufficient in one membrane, using only catalytic rules with the maximally parallel derivation mode, to generate any recursively enumerable set of natural numbers. Hence, by showing that P systems with purely catalytic rules working in the maximally parallel derivation mode can be considered as P systems working with the corresponding non-cooperative rules in the 1-restricted minimally parallel derivation mode when partitioning the rule sets for each membrane with respect to the catalysts, we obtain the astonishing result that in this case we get a characterization of the recursively enumerable sets of natural numbers by using only noncooperative rules, i.e., $NRE = O_* C_1\,(min_1)\,[\text{noncoop}]$.

4.1. Extended Spiking Neural P Systems

An *extended spiking neural P system* (of degree $m \geq 1$) (in the following we shall simply speak of an *ESNP system*) is a construct

$$\Pi = (m, S, R)$$

where

- m is the number of *neurons*; the neurons are uniquely identified by a number between 1 and m;
- S describes the *initial configuration* by assigning an initial value (of spikes) to each neuron;
- R is a finite set of *rules* of the form $\left(i, E/a^k \to P\right)$ such that $i \in [1..m]$ (specifying that this rule is assigned to neuron i), $E \subseteq REG(\{a\})$ is the *checking set* (the current number of spikes in the neuron has to be from E if this rule shall be executed), $k \in \mathbb{N}$ is the "number of spikes" (the energy) consumed by this rule, and P is a (possibly empty) set of *productions* of the form (l, a^w) where $l \in [1..m]$ (thus specifying the target neuron), $w \in \mathbb{N}$ is the *weight* of the energy sent along the axon from neuron i to neuron l.

A *configuration* of the ESNP system is described by specifying the actual number of spikes in every neuron. A *transition* from one configuration to another one is executed as follows: for each neuron i, we non-deterministically choose a rule $\left(i, E/a^k \to P\right)$ that can be applied, i.e., if the current value of spikes in neuron i is in E, neuron i "spikes", i.e., for every production (l, w) occurring in the set P we send w spikes along the axon from neuron i to neuron l. A *computation* is a sequence of configurations starting with the initial configuration given by S. An ESNP system can be used to generate sets from NRE (we do not distinguish between NRE and $RE(\{a\})$) as follows: a computation is called *successful* if it halts, i.e., if for no neuron, a rule can be activated; we then consider the contents, i.e., the number of spikes, of a specific neuron called *output neuron* in halting computations.

We now consider the ESNP system $\Pi = (m, S, R)$ as a network of cells $\Pi' = (m, \{a\}, S, R')$ working in the 1-restricted minimally parallel derivation mode, with

$$\begin{aligned} R' \;=\; & \left\{\left(E : \left(a^k, i\right) \to (a^{w_1}, l_1) \ldots (a^{w_n}, l_n)\right) \mid \right. \\ & \left.\left(i, E/a^k \to (l_1, a^{w_1}) \ldots (l_n, a^{w_n})\right) \in R\right\} \end{aligned}$$

and the partitioning R'_i, $1 \le i \le m$, of the rule set R' according to the set of neurons, i.e.,

$$\begin{aligned} R'_i \;=\; & \left\{\left(E : \left(a^k, i\right) \to (a^{w_1}, l_1) \ldots (a^{w_n}, l_n)\right) \mid \right. \\ & \left.\left(E : \left(a^k, i\right) \to (a^{w_1}, l_1) \ldots (a^{w_n}, l_n)\right) \in R'\right\}. \end{aligned}$$

The 1-restricted minimally parallel derivation mode chooses one rule – if possible – from every set R_i and then applies such a multiset of rules in parallel, which directly corresponds to applying one spiking rule in every neuron where a rule can be applied. Hence, it is easy to see that Π' and Π generate the same set from $RE\{a\}$ if in both systems we take the same cell/neuron for extracting the output. Due to the results valid for ESNP systems, see [1], we obtain

$$NRE = O_1C_3\,(min_1)\,[\text{ESNP}]\,.$$

4.2. Purely Catalytic P Systems

A noncooperative evolution rule is of the form $(I : (a, i) \to (y_1, 1) \dots (y_n, n))$ where a is a single symbol and I denotes the condition that is always fulfilled. A catalytic rule is of the form $(I : (c, i)\,(a, i) \to (c, i)\,(y_1, 1) \dots (y_n, n))$ where c is from a distinguished subset $V_C \subset V$ such that in all rules (noncooperative evolution rules, catalytic rules) of the whole system the y_i are from $(V - V_C)^*$ and the symbols a are from $(V - V_C)$. Imposing the restriction that the noncooperative evolution rules and the catalytic rules in a network of cells allow for finding a hierarchical tree structure of membranes such that symbols either stay in their membrane region or are sent out to the surrounding membrane region or sent into an inner membrane, then we get the classical catalytic P systems without priorities. Allowing regular sets checking for the non-appearance of specific symbols instead of I, we even get the original P systems with priorities. Catalytic P systems using only catalytic rules are called purely catalytic P systems. As we know from [6], only two (three) catalysts in one membrane are needed to obtain NRE with (purely) catalytic P systems without priorities working in the maximally parallel derivation mode, i.e., we can write these results as follows:

$$NRE = O_*C_1\,(max)\,[\mathrm{cat}_2] = O_*C_1\,(max)\,[\mathrm{pcat}_3]\,.$$

If we now partition the rule set in a purely catalytic P system according to the catalysts present in each membrane, this partitioning replaces the use of the catalysts when working in the 1-restricted minimally parallel derivation mode, because by definition from each of these sets then – if possible – exactly one rule (as with the use of the corresponding catalyst) is chosen: from the set of purely catalytic rules R we obtain the corresponding set of noncooperative rules R' as

$$\begin{aligned} R' &= \{(I : (a, i) \to (y_1, 1) \dots (y_n, n)) \mid \\ &\qquad (I : (c, i)\,(a, i) \to (c, i)\,(y_1, 1) \dots (y_n, n)) \in R\} \end{aligned}$$

as well as the corresponding partitioning of R' as

$$\begin{aligned} R'_{i,c} &= \{(I : (a, i) \to (y_1, 1) \dots (y_n, n)) \mid \\ &\qquad (I : (c, i)\,(a, i) \to (c, i)\,(y_1, 1) \dots (y_n, n)) \in R\}\,. \end{aligned}$$

Considering purely catalytic P systems in one membrane, we therefore infer the following quite astonishing result that when using the 1-restricted minimally parallel derivation mode for a suitable partitioning of rules we only need noncooperative rules:

$$NRE = O_*C_1\,(min_1)\,[\mathrm{noncoop}]\,.$$

5. Conclusions

The main purpose of this paper was to introduce the k-restricted minimally parallel derivation mode and to elaborate how this new derivation mode allows for capturing the main characteristics of well-known variants of membrane systems. The 1-restricted minimally parallel derivation mode, for example, allows us to interpret the way of how spiking neural P systems (without delays) work in a sequential way on the level of cells, but in the maximally parallel way on the level of the whole system by using this 1-restricted minimally parallel derivation mode for the whole system with a partitioning of the rules given by the individual neurons. P systems with purely catalytic rules working in the maximally parallel derivation mode can be considered as P systems working with the corresponding noncooperative rules in the 1-restricted minimally parallel derivation mode when partitioning the rule sets for each membrane with respect to the catalysts.

In the general framework considered in this paper, many other variants of static P systems and tissue P systems can be considered, hence, a great variety of such systems working in the k-restricted minimally parallel derivation mode, especially for $k = 1$, remains to be investigated in the future. Moreover, the basic k-restricted minimally parallel derivation mode may be restricted as exhibited for the other derivation modes as shown in [8], eventually with other variants of halting. As a specific example of new results of that kind we should like to mention that, with the variant of the k-restricted minimally parallel derivation mode having to include at most k rules from each set of the partitioning from which a rule is applicable to the current configuration into a multiset of rules to be applied, together with partial halting we can only obtain regular sets of natural numbers.

Acknowledgements

The authors gratefully acknowledge the useful suggestions and remarks from Artiom Alhazov and Markus Beyreder.

References

[1] ALHAZOV, A., FREUND, R., OSWALD, M., SLAVKOVIK, M., Extended spiking neural P systems generating strings and vectors of non-negative integers, in: H.J. Hoogeboom, Gh. Paun, G. Rozenberg (Eds.), Pre-proceedings of Membrane Computing, International Workshop, WMC7, Leiden, The Netherlands, 2006, 88–101.

[2] BERNARDINI, F., GHEORGHE, M., MARGENSTERN, M., VERLAN, S., Networks of cells and Petri nets, in: M. A. Gutiérrez-Naranjo, Gh. Păun, A. Romero-Jiménez, A. Riscos-Núñez (Eds.), Proc. Fifth Brainstorming Week on Membrane Computing, Sevilla, 2007, 33–62.

[3] CIOBANU, G., PAN, L., PĂUN, Gh., PÉREZ-JIMÉNEZ, M.J., P systems with minimal parallelism, Theoretical Computer Science 378 (1) (2007), 117–130.

[4] CSUHAJ-VARJÚ, E., Networks of language processors, Current Trends in Theoretical Computer Science (2001), 771–790.

[5] DASSOW, J., PĂUN, Gh., On the power of membrane computing, Journal of Universal Computer Science5 (2) (1999), 33–49.

[6] FREUND, R., KARI, L., OSWALD, M., SOSÍK, Computationally universal P systems without priorities: two catalysts are sufficient, Theoretical Computer Science 330 (2005), 251–266.

[7] FREUND, R., PĂUN, Gh., M.J. PÉREZ-JIMÉNEZ, Tissue-like P systems with channel states, Theoretical Computer Science 330 (2005), 101–116.

[8] FREUND, R., VERLAN, S., A formal framework for P systems, in: G. Eleftherakis, P. Kefalas, Gh. Paun (Eds.), Pre-proceedings of Membrane Computing, International Workshop – WMC8, Thessaloniki, Greece, 2007, 317–330.

[9] IONESCU, M., PĂUN, Gh., YOKOMORI, T., Spiking neural P systems, Fundamenta Informaticae 71, 2–3 (2006), 279–308.

[10] PĂUN, Gh., Computing with membranes, J. of Computer and System Sciences 61, 1 (2000), 108–143, and TUCS Research Report 208 (1998) (`http://www.tucs.fi`).

[11] PĂUN, Gh., Membrane Computing. An Introduction, Springer-Verlag, Berlin 2002.

[12] ROZENBERG, G., SALOMAA, A. (Eds.), Handbook of Formal Languages (3 volumes), Springer-Verlag, Berlin 1997.

[13] PĂUN, Gh., SAKAKIBARA, Y., YOKOMORI, T., P systems on graphs of restricted forms, Publicationes Matimaticae 60, 2002.

[14] PĂUN, Gh., YOKOMORI, T., Membrane computing based on splicing, in: E. Winfree and D. K. Gifford (Eds.), DNA Based Computers V, volume 54 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1999, 217–232.

[15] The P Systems Web Page: `http://ppage.psystems.eu`.

ONE-SIDED INSERTION AND DELETION: TRADITIONAL AND P SYSTEMS CASE

Alexander Krassovitskiy[1] Yurii Rogozhin[1,2] Sergey Verlan[2,3]
[1]Rovira i Virgili University,
Research Group on Mathematical Linguistics,
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
Email: alexander.krassovitskiy@estudiants.urv.cat
[2]Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
5, str. Academiei, MD-2028, Chişinău, Moldova
Email: rogozhin@math.md
[3] LACL, Département Informatique, Université Paris Est,
61, av. Général de Gaulle, 94010 Créteil, France
Email: verlan@univ-paris12.fr

Abstract
In this article we continue the investigation of insertion-deletion systems having a context only on one side of insertion or deletion rules. We investigate the combination of systems having minimal one-sided insertion and deletion rules and P systems and we show that the computational power is strictly increased. We also present a universality result for systems which insert or delete one symbol in two symbols asymmetrical insertion and deletion contexts. Finally we show that if deletion rules are not present, then some non-regular context-free languages may be obtained.

1. Introduction

The operations of insertion and deletion are fundamental in formal language theory, and generative mechanisms based on them were considered (with linguistic motivation) for some time, see [7] and [2]. Related formal language investigations can be found in several places; we mention only [3], [5], [9], [11]. In the last years, the study of these operations has received a new motivation from molecular computing, see [1], [4], [13], [15], [8].

In a general form, an insertion operation means adding a substring to a given string in a specified (left and right) context, while a deletion operation means removing a substring of a given string from a specified (left and right) context. A finite set of insertion-deletion rules, together with a set of axioms provide a language generating device (an InsDel system): starting from the set of initial strings and iterating insertion-deletion operations as defined by the given rules we get a language. The number of axioms, the length of the inserted or deleted strings, as well as the length of the contexts where these operations take place are natural descriptional complexity measures in this framework. As expected, insertion and deletion operations with context dependence are very powerful, leading to characterizations of recursively enumerable languages. Most of the papers mentioned above contain such results, in many cases improving the complexity of insertion-deletion systems previously available in the literature.

Some combinations of parameters lead to systems which are not computationally complete [10], [6] or even decidable [16]. It is worth to note that most of these results concern insertion-deletion systems having a context only on one side of insertion or deletion rules. Such kind of insertion-deletion systems can be investigated in combination with distributed computing concepts like P systems [12]. In this article we show that the distribution based on P systems increases the computational power with respect to a non-distributed insertion-deletion system. We also continue the investigation of the class of one-sided insertion-deletion systems and show a universality result for systems inserting and deleting one symbol in two-symbols asymmetrical context. Finally, we show that the class of minimal one-sided insertion-only systems contains non-regular context-free languages.

2. Prerequisites

All formal language notions and notations we use here are elementary and standard. The reader can consult any of the many monographs in this area – for instance, [14] – for the unexplained details.

We denote by $|w|$ the length of word w and by $card(A)$ the cardinality of the set A. An *InsDel system* is a construct $ID = (V, T, A, I, D)$, where V is an alphabet, $T \subseteq V$, A is a finite language over V, and I, D are finite sets of triples of the form (u, α, v), $\alpha \neq \varepsilon$ of strings over V, where ε denotes the empty string. The elements of T are *terminal* symbols (in contrast, those of $V - T$ are called nonterminals), those of A are *axioms*, the triples in I are *insertion rules*, and those from D are *deletion rules*. An insertion rule $(u, \alpha, v) \in I$ indicates that the string α can be inserted in between u and v, while a deletion rule $(u, \alpha, v) \in D$ indicates that α can be removed from the context (u, v). Stated otherwise, $(u, \alpha, v) \in I$ corresponds to the rewriting rule $uv \to u\alpha v$, and $(u, \alpha, v) \in D$ corresponds to the rewriting rule $u\alpha v \to uv$. We denote by $\Longrightarrow_{ins}$ the relation defined by an insertion rule (formally, $x \Longrightarrow_{ins} y$ iff $x = x_1uvx_2, y = x_1u\alpha vx_2$, for some $(u, \alpha, v) \in I$ and $x_1, x_2 \in V^*$) and by $\Longrightarrow_{del}$ the relation defined by a deletion rule (formally, $x \Longrightarrow_{del} y$ iff $x = x_1u\alpha vx_2, y = x_1uvx_2$, for some $(u, \alpha, v) \in D$ and $x_1, x_2 \in V^*$). We refer by $\Longrightarrow$ to union of the relations $\Longrightarrow_{ins}, \Longrightarrow_{del}$, and denote by $\Longrightarrow^*$ the reflexive and transitive closure of $\Longrightarrow$ (as usual, $\Longrightarrow^+$ is its transitive closure). The language generated by ID is defined by $L(ID) = \{w \in T^* \mid x \Longrightarrow^* w, \text{ for all } x \in A\}$.

The complexity of an InsDel system $ID = (V, T, A, I, D)$ is traditionally described by the vector $(n, m; p, q)$ called *weight*, where

$$\begin{aligned}
n &= \max\{|\alpha| \mid (u,\alpha,v) \in I\},\\
m &= \max\{|u| \mid (u,\alpha,v) \in I \text{ or } (v,\alpha,u) \in I\},\\
p &= \max\{|\alpha| \mid (u,\alpha,v) \in D\},\\
q &= \max\{|u| \mid (u,\alpha,v) \in D \text{ or } (v,\alpha,u) \in D\},
\end{aligned}$$

The *total weight* of ID is the sum $\gamma = m + n + p + q$.

However, it was shown in [16] that this complexity measure is not accurate and it cannot distinguish between universality and non-universality cases (there are families having same total weight but not the same computational power). In the same article it was proposed to use the length of each context instead of the maximum. More exactly,

$$\begin{aligned}
n &= \max\{|\alpha| \mid (u,\alpha,v) \in I\},\\
m &= \max\{|u| \mid (u,\alpha,v) \in I\},\\
m' &= \max\{|v| \mid (u,\alpha,v) \in I\},\\
p &= \max\{|\alpha| \mid (u,\alpha,v) \in D\},\\
q &= \max\{|u| \mid (u,\alpha,v) \in D\},\\
q' &= \max\{|v| \mid (u,\alpha,v) \in D\}.
\end{aligned}$$

Hence the complexity of an insertion-deletion system will be described by the vector $(n, m, m'; p, q, q')$ that we call *size*. We also denote by $INS_n^{m,m'}DEL_p^{q,q'}$ corresponding families of insertion-deletion systems. Moreover, we define the total weight of the system as the sum of all numbers above: $\psi = n+m+m'+p+q+q'$. Since it is known from [16] that systems using a context-free insertion or deletion of one symbol are not powerful, we additionally require $n + m + m' \geq 2$ and $p + q + q' \geq 2$.

If some of the parameters n, m, m', p, q, q' is not specified, then we write instead the symbol $*$. In particular, $INS_*^{0,0}DEL_*^{0,0}$ denotes the family of languages generated by *context-free InsDel systems*. If one of numbers from the couples m, m' and/or q, q' is equal to zero (while the other is not), then we say that corresponding families have a one-sided context.

InsDel systems of a “sufficiently large” weight can characterize RE, the family of recursively enumerable languages.

An *insertion-deletion P system* is the following construct:

$$\Pi = (V, T, \mu, M_1, \ldots, M_n, R_1, \ldots, R_n),$$

where

- V is a finite alphabet,
- $T \subseteq V$ is the terminal alphabet,
- μ is the membrane (tree) structure of the system which has n membranes (nodes). This structure will be represented by a word containing correctly nested marked parentheses.

- M_i, for each $1 \leq i \leq n$ is a finite language associated to the membrane i.
- R_i, for each $1 \leq i \leq n$ is a set of insertion and deletion rules with target indicators associated to membrane i and which have the following form: $(u, x, v; tar)_a$, where (u, x, v) is an insertion rule, and $(u, x, v; tar)_e$, where (u, x, v) is an deletion rule, and tar, called the *target indicator*, is from the set $\{here, in, out\}$.

Any m-tuple $(N_1, \ldots, N_n)$ of languages over V is called a configuration of Π. For two configurations $(N_1, \ldots, N_n)$ and $(N'_1, \ldots, N'_n)$ of Π we write $(N_1, \ldots, N_n) \Longrightarrow (N'_1, \ldots, N'_n)$ if we can pass from $(N_1, \ldots, N_n)$ to $(N'_1, \ldots, N'_m)$ by applying the insertion and deletion rules from each region of μ, in parallel, to all possible strings from the corresponding regions, and following the target indications associated with the rules. More specifically, if $w \in M_i$ and $r = (u, x, v; tar)_a \in R_i$, respectively $r = (u, x, v; tar)_a \in R_i$, such that $w \Longrightarrow^r_{ins} w'$, respectively $w \Longrightarrow^r_{del} w'$, then w' will go to the region indicated by tar. If $tar = here$, then the string remains in M_i, if $tar = out$, then the string is moved to the region immediately outside the membrane i (maybe, in this way the string leaves the system), if $tar = in$, then the string is moved to the region immediately below.Note that as strings are supposed to appear in arbitrary many copies, after the application of rule r in a membrane i the string w is still available in the same region.

A sequence of transitions between configurations of a given insertion-deletion P system Π, starting from the initial configuration $(M_1, \ldots, M_n)$, is called a computation with respect to Π. The result of a computation consists of all strings over T which are sent out of the system at any time during the computation. We denote by $L(\Pi)$ the language of all strings of this type. We say that $L(\Pi)$ is generated by Π. We denote by $ELSP_k(insdel, (n, m, m'; p, q, q'))$(see, for example [12]) the family of languages $L(\Pi)$ generated by insertion-deletion P systems of degree at most $k, k \geq 1$ having the size $(n, m, m'; p, q, q')$.

3. Results

In this section we present main results of the paper. We start with a result on computational completeness of one-sided insertion-deletion system of size $(1, 2, 0; 1, 0, 2)$. In order to proof the next theorem we recall the following lemma from [6]:

Lemma 1. *For any insertion-deletion system $ID = (V, T, A, I, D)$ with size $(n, m, m'; p, q, q')$ we can construct an insertion-deletion system $ID_2 = (V \cup \{X, Y\}, T, A_2, I_2, D_2 \cup D'_2)$ having the same size such that $L(ID_2) = L(ID)$. Moreover, all rules from I_2 have the form (u, α, v), where $|u| = m$, $|v| = m'$, all rules from D_2 have the form (u', α, v'), where $|u'| = q$, $|v'| = q'$ and $D'_2 = \{(\varepsilon, X, \varepsilon), (\varepsilon, Y, \varepsilon)\}$.*

Theorem 2. *$INS_1^{2,0}DEL_1^{0,2} = RE$.*

Proof. The proof of the theorem is based on a simulation of insertion-deletion systems of size $(1, 1, 1; 1, 1, 1)$. It is known that these systems generate any recursively enumerable language

[15]. Consider $ID = (V, T, A, I, D)$ to be such a system. Now we construct a system $ID_2 = (V_2, T, A, I_2, D_2)$ of size $(1, 2, 0; 1, 0, 2)$ that will generate the same language as ID.

Using Lemma 1 for normal form of insertion and deletion rules we may assume that rules have always two symbol in contexts, except for two deletion rules of the form $(\varepsilon, X, \varepsilon)$ and $(\varepsilon, Y, \varepsilon)$. It is clear that in order to show the inclusion $L(ID) \subseteq L(ID_2)$ it is sufficient to show how an insertion rule $(a, x, b) \in I$, with $a, b, x \in V$, may be simulated by using rules of system ID_2, *i.e.*, insertion rules of type $(a'b', x', \varepsilon)$ and deletion rules of type $(\varepsilon, y', b'c')$, with $a', b', c' \in V_2 \cup \{\varepsilon\}$, $x', y' \in V_2$. We may suppose that for any rule $(a, x, b) \in I$ it holds $x \neq b$. Indeed, if this is not the case then this rule may be replaced by two insertion rules (a, B, b), (a, b, B) and one deletion rule (b, B, b):

$$w_1abw_2 \Longrightarrow w_1aBbw_2 \Longrightarrow w_1abBbw_2 \Longrightarrow w_1abbw_2.$$

Similarly we may suppose that for a rule $(a, x, b) \in I$ it holds $x \neq a$.

Consider $V_2 = V \cup \{A_i, X_i, Y_i \mid 1 \leq i \leq card(I)\} \cup \{B_i \mid 1 \leq i \leq card(D)\}$. Let us label all rules from I by integer numbers. Consider now a rule $i : (a, x, b) \in I$, where $1 \leq i \leq card(I)$ is the label of the rule. We introduce insertion rules

$$(ab, Y_i, \varepsilon) \tag{1}$$
$$(a, A_i, \varepsilon) \tag{2}$$
$$(aA_i, x, \varepsilon) \tag{3}$$
$$(A_ix, X_i, \varepsilon) \tag{4}$$

and the deletion rules

$$(\varepsilon, A_i, xX_i) \tag{5}$$
$$(\varepsilon, X_i, bY_i) \tag{6}$$
$$(\varepsilon, Y_i, \varepsilon) \tag{7}$$

in D_2. The rule $i : (a, x, b) \in I$ is simulated as follows. We first perform insertions of symbols Y_i, A_i, x, and X_i:

$$\begin{aligned} w_1abw_2 &\Longrightarrow w_1abY_iw_2 \Longrightarrow w_1aA_ibY_iw_2 \\ &\Longrightarrow w_1aA_ixbY_iw_2 \Longrightarrow w_1aA_ixX_ibY_iw_2. \end{aligned}$$

and after that deletion of A_i, X_i, Y_i as follows:

$$w_1aA_ixX_ibY_iw_2 \Longrightarrow w_1axX_ibY_iw_2 \Longrightarrow w_1axbY_iw_2 \Longrightarrow w_1axbw_2.$$

In that way we simulate the rule $(a, x, b) \in I$ correctly. Now let us consider a deletion rule $(a, x, b) \in D$. Similarly as for insertion rule, we may suppose that for any rule $(a, x, b) \in D$ it holds $a \neq x$. If this is not the case then this rule may be replaced by two deletion rules (B, a, b), (a, B, b) and one insertion rule (a, B, a).

Let us label all rules from I by integer numbers. Consider now a rule $i : (a, x, b) \in D$, where $1 \leq i \leq card(D)$ is the label of the rule. We introduce insertion rule

$$(ax, B_i, \varepsilon) \tag{8}$$

and the deletion rules

$$(\varepsilon, x, B_i b) \tag{9}$$

$$(\varepsilon, B_i, b) \tag{10}$$

in D_2. The rule $i : (a, x, b) \in D$ is simulated as follows. We first perform insertions of B_i:

$$w_1 abw_2 \Longrightarrow^+ w_1 ax(B_i)^+ bw_2$$

and after that deletion of x and B_i (applicable to $w_1 axB_i bw_2$):

$$w_1 axB_i bw_2 \Longrightarrow w_1 aB_i bw_2 \Longrightarrow w_1 abw_2$$

Hence, $L(ID) \subseteq L(ID_2)$. Now in order to prove the converse inclusion, we consider the simulation of the rule $(a, x, b) \in I$. We observe that we perform insertion of non-terminal symbols Y_i, A_i, and X_i from V_2. After performing these insertions, the only way to get rid of these symbols is to perform the whole sequence of insertions and deletions. Indeed, in order to erase A_i at least one symbol X_i has to be inserted after x. And, in order to erase X_i at least one Y_i are needed after b. And, in case more than one symbol x are inserted, then it is impossible to eliminate the corresponding symbol A_i(as we assumed $x \neq b$). Now suppose that more than one symbol A_i, X_i or Y_i is inserted after insertion of x, then to finish the derivation we will have to eliminate every inserted A_i, X_i and Y_i. Moreover insertion of Y_i after ab insures that substring xb which appears after (3) can not be used by other rules until A_i, and X_i are removed.

Now consider the deletion rule (a, x, b). Here we insert B_i from V_2 corresponding to the deletion rule. It follows from (8) that insertion of B_i only possible if ax is on the left. Now we have two possible cases. The first case is to erase B_i straight after the insertion. It will remain the sentential form unchanged. The second case is to erase x by the rule (9) and after that erase B_i. In the case more than one B_i are inserted, we will have to remove every additional symbol B_i by the rule (10). And, since we supposed $a \neq x$, not more then one x can be deleted. It proves that $L(ID_2) \subseteq L(ID)$ and hence we have $L(ID_2) = L(ID)$. □

As a corollary we obtain the following result:

Corollary 3. $INS_1^{0,2}DEL_1^{2,0} = RE$.

Now we consider insertion-deletion system with rules having small context and with no deletion rules. In [13] it is already shown that the family $INS_n^{1,1}DEL_0^{0,0}$ is a subset of the family of context-free languages. Below we show that even a smaller subclass, $INS_1^{1,0}DEL_0^{0,0}$, having one-sided insertion rules contains non-regular context-free languages.

Theorem 4. $INS_1^{1,0}DEL_0^{0,0} \cap (CF \setminus REG) \neq \emptyset$

Proof. It is already known that $INS_1^{1,0}DEL_0^{0,0} \subseteq CF$, see [13]. To complete the assertion of the theorem it suffices to show that a non-regular context-free language might be generated by such systems.

Consider system $ID = (T, T, \{a\}, I, \emptyset)$, where $T = \{a, b, c, d\}$ and $I = \{(a, b, \varepsilon), (b, c, \varepsilon), (c, d, \varepsilon), (d, a, \varepsilon)\}$.

Let us denote by L' the language generated by ID ($L' = L(ID)$). Let us observe a possible derivation of ID (where S_x, $x \in T$ denotes a possible insertion site after the letter x):

$$\begin{aligned} aS_a &\Longrightarrow aS_abS_b \Longrightarrow^* aS_abS_bb^*S_b \Longrightarrow^* aS_ab^+cS_cb^*S_b \Longrightarrow^* \\ &aS_ab^+c^+S_cc^*S_cb^*S_b \Longrightarrow^* aS_ab^+c^+d^+S_dd^*S_dc^*S_cb^*S_b \Longrightarrow^* \\ &aS_ab^+c^+d^+S_daS_ad^*S_dc^*S_cb^*S_b \Longrightarrow \dots \end{aligned}$$

It is easy to observe that the language L' might be defined by the following recursive formulas:

$$\begin{aligned} L_1 &= b^* + b^+c^* + b^+c^+d^* \\ L_2 &= b^+c^+d^+(a(L_1 + L_2)^*)^*d^*c^*b^* \\ L' &= a(L_1 + L_2)^* \end{aligned}$$

Let $R = (abcd)^*(dcb)^*$. Consider the language $L'' = L' \cap R$. In this case we obtain $L'' = (abcd)^i(dcb)^j$, $j \leq i$. Indeed, the sequence dcb might be obtained if and only if an instance of L_2 is generated, *i.e.*, an instance of $abcd$ is present. Since in L_2 we obtain $d^*c^*b^*$, then the number of dcb in the intersection might be smaller than the number of $abcd$ obtained.

Now it is clear that $(abcd)^i(dcb)^j$, $j \leq i$ is a context-free language (by the inverse morphism $\{abcd \to x, dcb \to y$ it becomes the well known context-free language x^iy^j, $j \leq i$). Since the intersection of a context-free language with a regular language is a context-free language we obtain that L' is a context-free language. The assertion of the Theorem follows from this. □

Consider now insertion-deletion systems with rules of the size (1,1,0; 1,1,0). This class is not computationally complete, because it is a subclass of the class of systems with the size (1,1,0; 1,1,1) which is not computationally complete [6]. However, if rules of size $(1, 1, 0; 1, 1, 0)$ are used in the framework of P systems, then the computational power increases and it becomes possible to simulate any type-0 grammar.

Theorem 5. $ELSP_5(insdel, (1, 1, 0; 1, 1, 0)) = RE$.

Proof. Let $G = (N, T, S, R)$ be a type-0 grammar in Kuroda normal form. This means that all production rules in R are of the form:

$$AB \longrightarrow CD \text{ or}$$
$$A \longrightarrow BC \text{ or}$$
$$A \longrightarrow \alpha$$

where A, B, C and D are from N and $\alpha \in T \cup N \cup \{\varepsilon\}$. Suppose that rules in R are ordered and $n = card(R)$.

Now consider the following system

$$\Pi = (V, T, [_1\ [_2\ [_3\ [_4\ [_5\ ]_5]_4]_3]_2]_1, \{SX\}, \emptyset, \emptyset, \emptyset, \emptyset, R_1, R_2, R_3, R_4, R_5).$$

It has a new nonterminal alphabet $V = N \cup T \cup \{P_i^1, P_i^2 | i = 1, ..., n\} \cup \{X\}$. For every production $i : AB \longrightarrow CD$ from R with $A, B, C, D \in N$ we add following rules to R_1, R_2, R_3, R_4, and R_5 correspondingly:

$(\varepsilon, P_i^1, \varepsilon; in)_a$ to R_1;
$(P_i^1, A, \varepsilon; in)_e$ and $(\varepsilon, P_i^2, \varepsilon; out)_e$ to R_2;
$(P_i^1, B, \varepsilon; in)_e$ and $(P_i^2, C, \varepsilon; out)_a$ to R_3;
$(P_i^1, P_i^2, \varepsilon; in)_a$ and $(P_i^2, D, \varepsilon; out)_a$ to R_4;
$(\varepsilon, P_i^1, \varepsilon; out)_e$ to R_5.

For every production $i : A \longrightarrow BC$ from R where $A, B, C \in N$ we add rules:

$(\varepsilon, P_i^1, \varepsilon; in)_a$ to R_1;
$(P_i^1, A, \varepsilon; in)_e$ and $(\varepsilon, P_i^2, \varepsilon; out)_e$ to R_2;
$(P_i^1, X, \varepsilon; in)_a$ and $(P_i^2, B, \varepsilon; out)_a$ to R_3;
$(P_i^1, P_i^2, \varepsilon; in)_a$ and $(P_i^2, C, \varepsilon; out)_a$ to R_4;
$(\varepsilon, P_i^1, \varepsilon; out)_e$ to R_5.

For every production $i : A \longrightarrow \alpha$ from R where $A \in N, \alpha \in T \cup N$ we add rules:

$(\varepsilon, P_i^1, \varepsilon; in)_a$ to R_1;
$(P_i^1, A, \varepsilon; in)_e$ and $(\varepsilon, P_i^2, \varepsilon; out)_e$ to R_2;
$(P_i^1, P_i^2, \varepsilon; in)_a$ and $(P_i^2, \alpha, \varepsilon; out)_a$ to R_3;
$(\varepsilon, P_i^1, \varepsilon; out)_e$ to R_4.

For every production $i : A \longrightarrow \varepsilon$ from R with $A \in N$ we add rules $(\varepsilon, A, \varepsilon; here)_e$ to R_1.

Finally, we add to R_1 rules $(\varepsilon, X, \varepsilon; here)_e$ and $(\varepsilon, X, \varepsilon; out)_e$.

We claim that Π generates the same language as G. In fact it is enough to proof that every step in derivation by grammar G can be simulated in Π. Let us show as an example production $i : AB \longrightarrow CD \in R$. The simulation of this rule is controlled by symbols P_i^1 and P_i^2. Consider a string w_1ABw_2 in the skin region. We insert $P_i^1 : w_1ABw_2 \Longrightarrow w_1P_i^1ABw_2$ and send the obtained string to membrane 2. Here we delete $A : w_1P_i^1ABw_2 \Longrightarrow w_1P_i^1Bw_2$ and send the string to membrane 3. Next we delete $B : w_1P_i^1Bw_2 \Longrightarrow w_1P_i^1w_2$ and send the obtained string to membrane 4. Here we insert $P_i^2 : w_1P_i^1w_2 \Longrightarrow w_1P_i^1P_i^2w_2$ and send the string to 5. Now we remove $P_i^1 : w_1P_i^1P_i^2w_2 \Longrightarrow w_1P_i^2w_2$ pushing the string back to membrane 4. Next we insert $D : w_1P_i^2w_2 \Longrightarrow w_1P_i^2Dw_2$ and send the string to membrane 3. Next we insert $C : w_1P_i^2Dw_2 \Longrightarrow w_1P_i^2CDw_2$ is inserted and send the string to membrane 2. Finally we remove P_i^2 and gathering the previous steps we get $w_1ABw_2 \Longrightarrow^* w_1CDw_2$.

Productions of the form $A \longrightarrow BC$, where $A, B, C \in N$ can be simulated similarly. We use P_i^1 to control the sequence of rules to push sequential form "down" from R_1 to R_5, and P_i^2 to control the sequence of rules to push the sequential form "up" from R_5 to R_1. Clear, that we need even number of symbols in every production which is not the case for $A \longrightarrow BC$. One can mention that we use the symbol X just due to the definition of insertion-deletion systems restricts insertion or deletion rule (x, y, z) to have $y \neq \varepsilon$. In fact the context-free production $A \longrightarrow BC$ is simulated by two equivalent productions $A \longrightarrow XBC$ and $X \longrightarrow \varepsilon$ with a special nonterminal X. We use the same technique for productions $A \longrightarrow \alpha, A \in N, \alpha \in T \cup N$. The only difference is that we need only four membranes to make one symbol replacement.

According to the definition of insertion-deletion P systems the result of a computation consists of all strings over T which are sent out of the system at any time during the computation. This is formally provided by the rule $(\varepsilon, X, \varepsilon; out)_e$ in the skin membrane. This rule uses conventional notation from [12].

To claim the proof we observe that every correct sentential form has at most one symbol P_i^1 and $P_i^2, i = 1, ..., n$. And after insertion of P_i^1 in the skin membrane either all rules corresponding to i-th rule have to be applied (in the defined order) or the derivation is blocked. Hence, we have $L(G) = L(\Pi)$. □

Remark 6. We mention that in the proof above the simulation of productions have used embedded membranes to regulate the sequence of insertions and deletions. Similar result can be obtained if a matrix regulation for insertion-deletion systems is used.

Remark 7. Positions of symbols P_i^1 regarding symbols to be inserted and removed by i-th rule are determined by the form of rules for insertion and deletion (with left context so far). One can mention that the same result can be obtained for $ELSP_5(insdel, (1, 1, 0; 1, 0, 1))$ if contexts for insertion and deletion are on different sides. We leave it to be verified by scrupulous readers.

4. Conclusions

In this article we have investigated insertion-deletion systems with one-sided contexts. Computational complexity of one-sided insertion-deletion systems of the size (1,2,0;1,0,2) was given. Moreover, we showed that systems using only one-sided minimal insertion may generate non-regular context-free languages. Finally, we considered insertion-deletion rules of size $(1,1,0;1,1,0)$ in the framework of P systems and we showed that the computational power strictly increases. It is worth to note that this is the first example of such a relation, because all computationally complete insertion-deletion P systems present in the literature use such parameters for insertion-deletion rules that by themselves suffice to obtain the computational power of a Turing machine, without any distribution. The obtained result uses five membranes to achieve the computational completeness. We leave it an open question whether this number can be decreased for this kind of systems.

Acknowledgments

The first author acknowledges the grant of Ramon y Cajal from University Rovira i Virgili 2005/08 and grant no. MTM 63422 from the Ministry of Science and Education of Spain. The second author acknowledges the support of European Commission, project MolCIP, MIF1-CT-2006-021666. The second and the third author acknowledge the Science and Technology Center in Ukraine, project 4032.

References

[1] DALEY, M., KARI, L., GLOOR, G., SIROMONEY, R., Circular contextual insertions/deletions with applications to biomolecular computation, in: Proc. of 6th Int. Symp. on String Processing and Information Retrieval, SPIRE'99, Cancun, Mexico, 1999, 47–54.

[2] GALIUKSCHOV, B., Semicontextual grammars, Matematika Logica i Matematika Linguistika (in Russian), Tallin University, 1981, 38–50.

[3] KARI, L., On Insertion and Deletion in Formal Languages, PhD Thesis, University of Turku, 1991.

[4] KARI, L., PĂUN, Gh., THIERRIN, G, YU, S., At the crossroads of DNA computing and formal languages: characterizing RE using insertion-deletion systems, in: Proc. of 3rd DIMACS Workshop on DNA Based Computing, Philadelphia, 1997, 318–333.

[5] KARI, L., THIERRIN, G., Contextual insertion/deletion and computability, Information and Computation 131 (1) (1996), 47–61.

[6] KRASSOVITSKIY, A., ROGOZHIN, Yu., VERLAN, S., Further results on insertion-deletion systems with one-sided contexts, Pre-proceedings of the 2nd International Conference on Language and Automata, Theory and Application, LATA 2008, March 13-19, 2008, Technical Reports of Research Group on Mathematical Linguistics, No. 36/08, 2008, 347–358.

[7] MARCUS, S., Contextual grammars, Rev. Roum. Math. Pures Appl. 14 (1969), 1525–1534.

[8] MARGENSTERN, M., PĂUN, Gh., ROGOZHIN, Yu., VERLAN, S., Context-free insertion-deletion systems, Theoretical Computer Science 330 (2005), 339–348.

[9] MARTÍN-VIDE, C., PĂUN, Gh., SALOMAA, A., Characterizations of recursively enumerable languages by means of insertion grammars, Theoretical Computer Science 205 (1–2) (1998), 195–205.

[10] MATVEEVICI, A., ROGOZHIN, Yu., VERLAN, S., Insertion-deletion systems with one-sided contexts, Lecture Notes in Computer Science 4664 , Springer, 2007, 205–217.

[11] PĂUN, Gh., Marcus Contextual Grammars, Kluwer, Dordrecht, 1997.

[12] PĂUN, Gh., Membrane Computing. An Introduction, Springer-Verlag, Berlin, 2002, 163, 226–230.

[13] PĂUN, Gh., ROZENBERG, G., SALOMAA, A., DNA Computing. New Computing Paradigms, Springer-Verlag, Berlin, 1998.

[14] ROZENBERG, G., SALOMAA, A. (Eds.), Handbook of Formal Languages, Springer-Verlag, Berlin, 1997.

[15] TAKAHARA, A., YOKOMORI, T., On the computational power of insertion-deletion systems, in: Proc. of 8th International Workshop on DNA-Based Computers, DNA8, Sapporo, Japan, June 10–13, 2002, Revised Papers, Lecture Notes in Computer Science 2568, 2003, 269–280.

[16] VERLAN, S., On minimal context-free insertion-deletion systems, Journal of Automata, Languages and Combinatorics 12 (1/2) (2007), 317-328.

A SMALL UNIVERSAL SPIKING NEURAL P SYSTEM

Turlough Neary
Boole Centre for Research in Informatics
University College Cork, Ireland
Email: tneary@cs.may.ie

Abstract
In this work we give a small extended spiking neural P system that is weakly universal. This system is significantly smaller than the smallest strongly universal spiking neural P systems. Strong universality has strict conditions regarding the encoding of input and decoding of output. Weak universality has more relaxed conditions regarding the encoding of input and decoding of output. Păun and Păun [10] gave a strongly universal spiking neural P system with 84 neurons and another that has extended rules with 49 neurons. Subsequently, the number of neurons used for strong universality was reduced from 84 to 67 and from 49 to 41 by Zhang et al. [11]. Here we give a weakly universal spiking neural P system that uses extended rules and has only 12 neurons.

1. Introduction

Spiking neural P systems [2] are quite a new computational model that are a synergy inspired by P systems and spiking neural networks. It has been shown that these systems are computationally universal [2]. Before we discuss results in the area of small universal spiking neural P systems, we note the two different notions of universality given by Korec [5]. Strong universality has strict conditions regarding the encoding of input and decoding of output. Weak universality has more relaxed conditions regarding the encoding of input and decoding of output. Recently, Păun and Păun [10] gave two small strongly universal spiking neural P systems; A spiking neural P system with 84 neurons and an extended spiking neural P system with 49 neurons (and without delay). Păun and Păun conjectured that it is not possible to give a significant decrease in the number of neurons of their two universal systems. Zhang et al. [11] offered such a significant decrease in the number of neurons used to give such small universal systems. They give a strongly universal spiking neural P system with 67 neurons and another, which has extended rules (without delay), with 41 neurons. Here we give an extended spiking neural P system with 12 neurons that is weakly universal and also uses rules without delay.

From a previous result [9] it is known that there exists no universal spiking neural P system that simulates Turing machines in less the exponential time and space. It is a relatively straightforward matter to generalise this result to show that extended spiking neural P systems suffer from the same inefficiencies. It immediately follows that the universal system we present here and those found in [10, 11] have exponential time and space requirements. However, it is possible to give a time efficient spiking neural P system when we allow exhaustive use of rules. A universal extended spiking neural P system with exhaustive use of rules has been given that simulates Turing machines in polynomial time [9]. Furthermore, this system has only 18 neurons. Spiking neural P systems with exhaustive use of rules were originally proved computationally universal by Ionescu et al. [3]. However, the technique used to prove universality suffered from an exponential time overhead.

Using different forms of spiking neural P systems, a number of time efficient (polynomial or constant time) solutions to NP-hard problems have been given [1, 6, 7]. All of these solutions to NP-hard problems rely on families of spiking neural P systems. Specifically, the size of the problem instance determines the number of neurons in the spiking neural P system that solves that particular instance. This is similar to solving problems with circuits families where each input size has a specific circuit that solves it. Ionescu and Sburlan [4] have shown that spiking neural P systems simulate circuits in linear time.

In the next two sections we give definitions for spiking neural P systems and register machines and explain the operation of both. Following this, in Section 4 we give an extended spiking neural P system with 12 neurons that is weakly universal and uses rules without delay.

2. Spiking neural P system

Definition 1 (Spiking neural P systems).
A spiking neural P system is a tuple $\Pi = (O, \sigma_1, \sigma_2, \cdots, \sigma_m, syn, in, out)$*, where:*

1. $O = \{s\}$ *is the unary alphabet (*s *is known as a spike),*
2. $\sigma_1, \sigma_2, \cdots, \sigma_m$ *are neurons, of the form* $\sigma_i = (n_i, R_i), 1 \leqslant i \leqslant m$*, where:*
   (a) $n_i \geqslant 0$ *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. $E/s^b \rightarrow s; d$*, where* E *is a regular expression over* s*,* $b \geqslant 1$ *and* $d \geqslant 0$*,*
      ii. $s^e \rightarrow \lambda$*, where* λ *is the empty word,* $e \geqslant 1$*, and for all* $E/s^b \rightarrow s; d$ *from* R_i $s^e \notin L(E)$ *where* $L(E)$ *is the language defined by* E*,*
3. $syn \subseteq \{1, 2, \cdots, m\} \times \{1, 2, \cdots, m\}$ *is the set of synapses between neurons, where* $i \neq j$ *for all* $(i, j) \in syn$*,*
4. $in, out \in \{\sigma_1, \sigma_2, \cdots, \sigma_m\}$ *are the input and output neurons, respectively.*

In the same manner as in [10], spikes are introduced into the system from the environment by reading in a binary sequence (or word) $w \in \{0, 1\}$ via the input neuron σ_1. The sequence w is read from left to right one symbol at each timestep. If the read symbol is 1 then a spike enters the input neuron on that timestep.

A firing rule $r = E/s^b \rightarrow s; d$ is applicable in a neuron σ_i if there are $j \geqslant b$ spikes in σ_i and $s^j \in L(E)$ where $L(E)$ is the set of words defined by the regular expression E. If, at time t, rule r is executed then b spikes are removed from the neuron, and at time $t+d$ the neuron fires. When a neuron σ_i fires a spike is sent to each neuron σ_j for every synapse (i,j) in Π. Also, the neuron σ_i remains closed and does not receive spikes until time $t+d$ and no other rule may execute in σ_i until time $t+d+1$. A forgeting rule $r' = s^e \rightarrow \lambda$ is applicable in a neuron σ_i if there are exactly e spikes in σ_i. If r' is executed then e spikes are removed from the neuron. At each timestep t a rule must be applied in each neuron if there is one or more applicable rules at time t. Thus, while the application of rules in each individual neuron is sequential the neurons operate in parallel with each other.

Note from 2b(i) of Definition 1 that there may be two rules of the form $E/s^b \rightarrow s; d$, that are applicable in a single neuron at a given time. If this is the case then the next rule to execute is chosen non-deterministically. The output is the time between the first and second spike in the output neuron.

An extended spiking neural P system [10] has more general rules of the form $E/s^b \rightarrow s^p; d$, where $b \geqslant p \geqslant 1$. Thus, a synapse in a spiking neural P system with extended rules may transmit more than one spike in a single timestep.

3. Register machines and notions of universality

Definition 2 (Register machine)**.** *A register machine is a tuple $C = (z, r_1, r_m, Q, q_1, q_h)$, where z gives the number of registers, r_1 and r_m are the input and output registers respectively, $Q = \{q_1, q_2, \cdots, q_h\}$ is the set of instructions, $q_1, q_h \in Q$ are the initial and halt instructions, respectively.*

Each register r_j stores a natural number value $x \geqslant 0$. Each instruction q_i is of one of the following two forms $q_i : INC(j)$ or $q_i : DEC(j)q_k$, and is executed as follows:

- $q_i : INC(j)$ increment the value x stored in register r_j by 1 and move to instruction q_{i+1}.
- $q_i : DEC(j)q_k$ if the value x stored in register r_j is greater than 0 then decrement this value by 1 and move to instruction q_{i+1}, otherwise if $x = 0$ move to instruction q_k.

At the beginning of a computation the first instruction executed is q_1. The input to the register machine is initially stored in register r_1. If the register machine's control enters instruction q_h then the computation halts at that timestep. The result of the computation is the value x stored in the output register r_m when the computation halts.

In Section 14.1 of his book, Minsky [8] proves that register machines with only two registers are universal.

Theorem 1 (Minsky [8]). *For any Turing machine T there exists a program machine M_T with just two registers that behaves the same as T (in the sense described in sections 10.1 and 11.2) when started with zero in one register and $2^a 3^m 5^n$ in the other. This machines uses only the operations $\boxed{'}$ and $\boxed{-}$, assuming that the successor instruction contains the "go" information for the next instruction.*

Minsky refers to register machines as program machines (i.e. M_T satisfies Definition 2). The operations $\boxed{'}$ and $\boxed{-}$ are identical to the instructions INC and DEC, which we defined above. The term "behaves the same as T" is well defined in Minsky's book and the encoding and decoding used by M_T satisfy Korec's notion of weak universality.

Recall that the output of a spiking neural P system Π is the time interval between the first and second spike. If the binary sequence $10^{y-1}1$ is given as input to Π, then the output of the computation is given by $\Pi(y)$. Let $(\phi_0, \phi_1, \phi_2, \ldots)$ be a Gödel enumeration of all unary partial recursive functions. Then we say that a spiking neural P system Π_U is weakly universal if $\phi_x(y) = f(\Pi_U(g(x,y))$ where g and f are recursive functions.

The small universal spiking neural P systems of Păun and Păun [10] and of Zhang et al. [11] simulate the 22 instruction strongly universal register machine of Korec [5]. In addition these spiking neural P systems satisfy the strict encoding and decoding requirements of Korec's [5] notion of strong universality. For weak universality it is sufficient to have encoding and decoding functions that are recursive. For this reason we note only that both the encoding ($2^a 3^m 5^n$) and decoding functions for M_T in Theorem 1 are recursive. It is not necessary for our purposes to discuss the encoding and decoding for M_T in more detail. The extended spiking neural P systems we give in the next section simulates a weakly universal 2 register machine. Thus, our system is also weakly universal.

4. A small universal spiking neural P system

In this section we give our small universal spiking neural P system. We prove the universality of our system by showing that it simulates a weakly universal register machine C_2 that has only two registers. Using Minsky's proof of Theorem 1 finding such a register machine is relatively straightfroward. As noted at the end of the previous section the encoding and decoding for such register machines are recursive and thus it is not necessary to concern ourselves with other details of C_2 here.

Theorem 2. *Let C_2 be a weakly universal register machine with 2 registers. Then there is a weakly universal extended spiking neural P system Π_{C_2} that simulates the computation of C_2 and has only 12 neurons.*

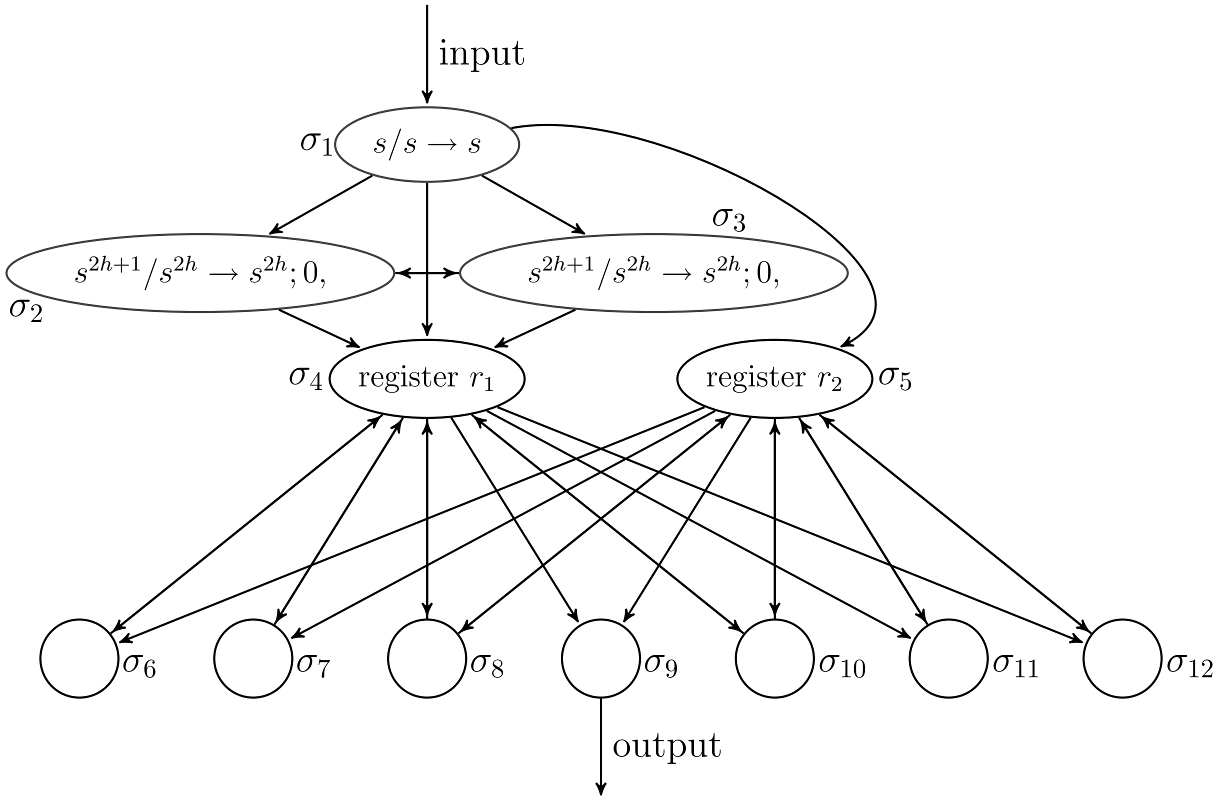


Figure 1: Universal extended spiking neural P system Π_{C_2}.

Proof. Let $C_2 = (2, r_1, r_1, Q, q_1, q_h)$ where $Q = \{q_1, q_2, \cdots, q_h\}$. Our spiking neural P system Π_{C_2} is given by Figure 1, and Tables 1 and 2. The algorithm given for Π_{C_2} is deterministic.

Encoding of a configuration of C_2 and reading in input to Π_{C_2}. A configuration of C_2 is stored as spikes in the neurons of Π_{C_2}. The next instruction q_i to be executed is stored in each of the neurons σ_4 and σ_5 as $2(h+i)$ spikes. Let x_1 and x_2 be the values stored in registers r_1 and r_2, respectively. Then x_1 and x_2 are stored as $4hx_1$ and $4hx_2$ spikes in neurons σ_4 and σ_5, respectively.

The input to Π_{C_2} is read into the system via the input neuron σ_1 as shown in Figure 1. If the input to C_2 is x then the binary sequence $w = 10^{x-1}1$ is read in via the input neuron σ_1. In Figure 1 σ_2, σ_3, σ_4, and σ_5 initially contain $2h$ spikes before the computation begins. Neurons σ_2 and σ_3 receive the first spike from σ_1 at time t_1 and the second spike at time t_{x+1}. Thus, σ_2 and σ_3 fire on every timestep between times t_1 and t_{x+2} and send a total of $4hx$ spikes to σ_4. Therefore, when σ_1 fires for the second time, after $x+2$ timesteps neuron σ_4 contains $4hx + 2(h+1)$ spikes, the encoding $(4hx)$ of the input x and the encoding $(2(h+1))$ of the initial instruction q_1. Note that at time t_{x+2} neuron σ_5 also contains the encoding $(2(h+1))$ of the initial instruction q_1.

neuron	rules
σ_1	$s/s \rightarrow s;0$
σ_2,σ_3	$s^{2h+1}/s^{2h} \rightarrow s^{2h};0$
σ_4	$s^{2(h+i)}(s^{4h})^*/s^{2(h+i)} \rightarrow s^{2(h+i)-1};0$ if $q_i : INC(1) \in \{Q\}$
	$s^{4h+2(h+i)}(s^{4h})^*/s^{4h+2(h+i)} \rightarrow s^{2(h+i)};0$ if $q_i : DEC(1) \in \{Q\}$
	$s^{2(h+i)}/s^{2(h+i)} \rightarrow s^{2(h+i)-1};0$ if $q_i : DEC(1) \in \{Q\}$
	$s^{2(h+i)}(s^{4h})^*/s^{2(h+i)} \rightarrow s;0$ if $q_i : INC(2) \in \{Q\}$ or $q_i : DEC(2) \in \{Q\}$
	$s^{6h+3}(s^{4h})^*/s^{6h} \rightarrow s;0,\quad s^{2h+3} \rightarrow \lambda,\quad s^{7}(s^{4h})^*/s^{4h} \rightarrow s^{2h};0,\quad s^3/s^3 \rightarrow s;0,$
σ_5	$s^{2(h+i)}(s^{4h})^*/s^{2(h+i)} \rightarrow s^{2(h+i)-1};0$ if $q_i : INC(2) \in \{Q\}$
	$s^{4h+2(h+i)}(s^{4h})^*/s^{4h+2(h+i)} \rightarrow s^{2(h+i)};0$ if $q_i : DEC(2) \in \{Q\}$
	$s^{2(h+i)}/s^{2(h+i)} \rightarrow s^{2(h+i)-1};0$ if $q_i : DEC(2) \in \{Q\}$
	$s^{2(h+i)}(s^{4h})^*/s^{2(h+i)} \rightarrow s;0$ if $q_i : INC(1) \in \{Q\}$ or $q_i : DEC(1) \in \{Q\}$

Table 1: This table gives the rules in each of the neurons σ_1 to σ_5 of Π_{C_2}.

$\mathbf{\Pi_{C_2}}$ **simulating** $\mathbf{q_i : INC(1)}$. Let there be x_1 spikes in register r_1 and x_2 spikes in register r_2. Then the simulation of $q_i : INC(1)$ begins at time t_j with $4hx_1 + 2(h+i)$ spikes in σ_4 and $4hx_2 + 2(h+i)$ spikes in σ_5. We explain the simulation by giving the number of spikes in each neuron and the rule that is to be applied in each neuron at time t. For example at time t_j we have

$$\begin{aligned} t_j : \sigma_4 &= 4hx_1 + 2(h+i), & s^{2(h+i)}(s^{4h})^*/s^{2(h+i)} &\rightarrow s^{2(h+i)-1};0,\\ \sigma_5 &= 4hx_2 + 2(h+i), & s^{2(h+i)}(s^{4h})^*/s^{2(h+i)} &\rightarrow s;0. \end{aligned}$$

where on the left $\sigma_k = y$ gives the number y of spikes in neuron σ_k at time t_j and on the right is the next rule that is to be applied at time t_j if there is an applicable rule at that time. Thus, from Figure 1, when we apply the rule $s^{2(h+i)}(s^{4h})^*/s^{2(h+i)} \rightarrow s^{2(h+i)-1};0$ in neuron σ_4 and the rule $s^{2(h+i)}(s^{4h})^*/s^{2(h+i)} \rightarrow s;0$ in σ_5 at time t_j we get

$$\begin{aligned} t_{j+1} : \sigma_4 &= 4hx_1, & &\\ \sigma_5 &= 4hx_2, & &\\ \sigma_6,\sigma_7,\sigma_8 &= 2(h+i), & s^{2(h+i)}/s^{2(h+i)} &\rightarrow s^{2h};0,\\ \sigma_9,\sigma_{11},\sigma_{12} &= 2(h+i), & s^{2(h+i)} &\rightarrow \lambda,\\ \sigma_{10} &= 2(h+i), & s^{2(h+i)}/s^{2(h+i)} &\rightarrow s^{2(i+1)};0, \end{aligned}$$

$$\begin{aligned} t_{j+2} : \sigma_4 &= 4h(x_1+1) + 2(h+i+1),\\ \sigma_5 &= 4hx_2 + 2(h+i+1). \end{aligned}$$

At time t_{j+2} the simulation of $q_i : INC(1)$ is complete. The encoded register value has been

neuron	rules
σ_6, σ_7	$s^{2(h+i)}/s^{2(h+i)} \rightarrow s^{2h}; 0$ if $q_i : INC(1) \in \{Q\}$
	$s^{2(h+i)} \rightarrow \lambda$ if $q_i : INC(1) \notin \{Q\}$
	$s^{2(h+i)+1} \rightarrow \lambda, \quad s^{2h} \rightarrow \lambda, \quad s \rightarrow \lambda$
σ_8	$s^{2(h+i)}/s^{2(h+i)} \rightarrow s^{2h}; 0, \quad s^{2(h+i)+1}/s^{2(h+i)+1} \rightarrow s^{2h}; 0, \quad s^{2h} \rightarrow \lambda, \quad s \rightarrow \lambda$
σ_9	$s^{2(h+i)} \rightarrow \lambda, \quad s^{2(h+i)+1} \rightarrow \lambda, \quad s^{2h} \rightarrow \lambda, \quad s/s \rightarrow s; 0$
σ_{10}	$s^{2(h+i)}/s^{2(h+i)} \rightarrow s^{2(i+1)}; 0$ if $q_i : INC \in \{Q\}$ and $q_{i+1} \neq q_h$
	$s^{2(h+i)}/s^{2(h+i)} \rightarrow s^3; 0$ if $q_i : INC \in \{Q\}$ and $q_{i+1} = q_h$
	$s^{2(h+i)+1}/s^{2(h+i)+1} \rightarrow s^{2(i+1)}; 0$ if $q_i : DEC \in \{Q\}$ and $q_{i+1} \neq q_h$
	$s^{2(h+i)+1}/s^{2(h+i)+1} \rightarrow s^3; 0$ if $q_i : DEC \in \{Q\}$ and $q_{i+1} = q_h$
	$s^{2(h+i)}/s^{2(h+i)} \rightarrow s^{2k}; 0$ if $q_i : DECq_k \in \{Q\}$ and $q_k \neq q_h$
	$s^{2(h+i)}/s^{2(h+i)} \rightarrow s^3; 0$ if $q_i : DECq_k \in \{Q\}$ and $q_k = q_h$
	$s^{2h} \rightarrow \lambda, \quad s \rightarrow \lambda$
σ_{11}, σ_{12}	$s^{2(h+i)}/s^{2(h+i)} \rightarrow s^{2h}; 0$ if $q_i : INC(2) \in \{Q\}$
	$s^{2(h+i)} \rightarrow \lambda$ if $q_i : INC(2) \notin \{Q\}$
	$s^{2(h+i)+1} \rightarrow \lambda, \quad s^{2h} \rightarrow \lambda, \quad s \rightarrow \lambda$

Table 2: This table gives the rules in each of the neurons σ_6 to σ_{12} of Π_{C_2}.

incremented by increasing it from $4hx_1$ to $4h(x_1+1)$. The encoding $2(h+i+1)$ of the next instruction q_{i+1} has been established.

Π_{C_2} simulating $\mathbf{q_i : DEC(1)q_k}$. As above the simulation begins at time t_j giving

$$t_j : \sigma_4 = 4hx_1 + 2(h+i), \qquad s^{4h+2(h+i)}(s^{4h})^*/s^{4h+2(h+i)} \rightarrow s^{2(h+i)}; 0,$$
$$\sigma_5 = 4hx_2 + 2(h+i), \qquad s^{2(h+i)}(s^{4h})^*/s^{2(h+i)} \rightarrow s; 0,$$

$$t_{j+1} : \sigma_4 = 4h(x_1-1),$$
$$\sigma_5 = 4hx_2,$$
$$\sigma_6, \sigma_7, \sigma_9, \sigma_{11}, \sigma_{12} = 2(h+i)+1, \qquad s^{2(h+i)+1} \rightarrow \lambda,$$
$$\sigma_8 = 2(h+i)+1, \qquad s^{2(h+i)+1}/s^{2(h+i)+1} \rightarrow s^{2h}; 0,$$
$$\sigma_{10} = 2(h+i)+1, \qquad s^{2(h+i)+1}/s^{2(h+i)+1} \rightarrow s^{2(i+1)}; 0,$$

$$t_{j+2} : \sigma_4 = 4h(x_1-1) + 2(h+i+1),$$
$$\sigma_5 = 4hx_2 + 2(h+i+1).$$

At time t_{j+2} the simulation of $q_i : DEC(1)q_k$ is complete. The encoded register value has been decremented by decreasing it from $4hx_1$ to $4h(x_1-1)$. The encoding $2(h+i+1)$ of the next instruction q_{i+1} has been established. In the above example we assume that register r_1 has value $x_1>0$. If $x_1=0$ then we get the following

$$\begin{aligned}
t_j : \sigma_4 &= 2(h+i), && s^{2(h+i)}/s^{2(h+i)} \rightarrow s^{2(h+i)-1};0,\\
\sigma_5 &= 4hx_2+2(h+i), && s^{2(h+i)}(s^{4h})^*/s^{2(h+i)} \rightarrow s;0,\\[1em]
t_{j+1} : \sigma_5 &= 4hx_2, && \\
\sigma_6,\sigma_7,\sigma_9,\sigma_{11},\sigma_{12} &= 2(h+i), && s^{2(h+i)} \rightarrow \lambda,\\
\sigma_8 &= 2(h+i), && s^{2(h+i)}/s^{2(h+i)} \rightarrow s^{2h};0,\\
\sigma_{10} &= 2(h+i), && s^{2(h+i)}/s^{2(h+i)} \rightarrow s^{2k};0,\\[1em]
t_{j+2} : \sigma_4 &= 2(h+k), && \\
\sigma_5 &= 4hx_2+2(h+k). &&
\end{aligned}$$

Note that at time t_{j+2}, when the simulation is complete, the encoding $2(h+k)$ of the next instruction q_{i+1} has been established.

Halting. The halt instruction q_h is encoded as $2h+3$ spikes. Thus, if C_2 enters the halt instruction q_h we get the following

$$\begin{aligned}
t_j : \sigma_4 &= 4hx_1+2h+3, && s^{6h+3}(s^{4h})^*/s^{6h} \rightarrow s;0,\\
\sigma_5 &= 4hx_2+2h+3, && \\[1em]
t_{j+1} : \sigma_4 &= 4h(x_1-1)+3, && s^{7}(s^{4h})^*/s^{4h} \rightarrow s^{2h};0,\\
\sigma_5 &= 4hx_2+2h+3, && \\
\sigma_6,\sigma_7,\sigma_8,\sigma_{10},\sigma_{11},\sigma_{12} &= 1, && s \rightarrow \lambda,\\
\sigma_9 &= 1, && s/s \rightarrow s;0,\\[1em]
t_{j+2} : \sigma_4 &= 4h(x_1-2)+3, && s^{7}(s^{4h})^*/s^{4h} \rightarrow s^{2h};0,\\
\sigma_5 &= 4hx_2+2h+3, && \\
\sigma_6,\sigma_7,\sigma_8,\sigma_9,\sigma_{10},\sigma_{11},\sigma_{12} &= 2h, && s^{2h} \rightarrow \lambda.
\end{aligned}$$

The rule $s^7(s^{4h})^*/s^{4h} \rightarrow s^{2h};0$, is applied a further $x_1 - 2$ times in σ_9 until we get

$$\begin{aligned} t_{j+x_1} : \sigma_4 &= 3, && s^3/s^3 \rightarrow s;0,\\ \sigma_5 &= 4hx_2 + 2h + 3, &&\\ \sigma_6, \sigma_7, \sigma_8, \sigma_9, \sigma_{10}, \sigma_{11}, \sigma_{12} &= 2h, && s^{2h} \rightarrow \lambda, \end{aligned}$$

$$\begin{aligned} t_{j+x_1+1} : \sigma_5 &= 4hx_2 + 2h + 3, &&\\ \sigma_6, \sigma_7, \sigma_8, \sigma_{10}, \sigma_{11}, \sigma_{12} &= 1, && s \rightarrow \lambda,\\ \sigma_9 &= 1, && s/s \rightarrow s;0. \end{aligned}$$

As usual the output is the time interval between the first and second spikes that are sent out of the output neuron. Note from above that the output neuron σ_9 fires for the first time at timestep t_{j+1} and for the second time at timestep t_{j+x_1+1}. Thus, the output of Π_{C_2} is x_1 the value of the output register r_1 when C_2 enters the halt instruction q_h. Note that if $x_1 = 0$ then the rule $s^{2h+3} \rightarrow \lambda$ is executed at timestep t_j and thus no spikes will be sent out of the output neuron.

We have shown how to simulate arbitrary instructions of the form $q_i : INC(1)$ and $q_i : DEC(1)q_k$. Instructions of the form $q_i : INC(2)$ and $q_i : DEC(2)q_k$, which operate on register r_2, are simulated in a similar manner Immediately following the simulation of an instruction Π_{C_2} is configured to simulate the next instruction. Thus, Π_{C_2} simulates the computation of C_2. □

The algorithm used by Π_{C_2} could be easily modified to simulate strongly universal register machines thus giving small extended spiking neural P systems that are strongly universal. Each additional register would require an extra three neurons. Also, if we wish to simulate a register machine that has two input registers we would require a further three neurons.

The reachability question for spiking neural P systems is as follows; Given a configuration c_x of a spiking neural P systems does it ever enter a configuration c_y. It is worth noting that using the results in Theorem 2 smaller spiking neural P systems with undecidable reachability questions may be given. Such systems may be given by removing the output neurons and the neurons for initialising the system (the input module) from Π_{C_2}. Thus, there exist spiking neural P systems with 8 neurons which have undecidable reachability questions.

Acknowledgements

The author would like to thank the anonymous reviewers and Rudolf Freund for their careful reading of the paper, and for their helpful suggestions and comments. The author is funded by Science Foundation Ireland Research Frontiers Programme grant number 07/RFP/CSMF641.

References

[1] CHEN, H., IONESCU, M., ISHDORJE, T., On the efficiency of spiking neural P systems, in: M. A. Gutiérrez-Naranjo et al. (Eds.), Proceedings of Fourth Brainstorming Week on Membrane Computing, Sevilla, Feb. 2006, 195–206, .

[2] IONESCU, M., PĂUN, Gh., YOKOMORI, T., Spiking neural P systems, Fundamenta Informaticae 71(2-3) (2006), 279–308.

[3] IONESCU, M., PĂUN, Gh., YOKOMORI, T., Spiking neural P systems with exhaustive use of rules, International Journal of Unconventional Computing 3 (2) (2007), 135–153.

[4] IONESCU, M., SBURLAN, D., Some applications of spiking neural P systems, in: George Eleftherakis et al. (Eds.), Proceedings of the Eightth Workshop on Membrane Computing, Thessaloniki, June 2007, 383–394.

[5] KOREC, I., Small universal register machines, Theoretical Computer Science 168 (2) (1996), 267–301.

[6] LEPORATI, A., ZANDRON, C., FERRETTI, C., MAURI, G., On the computational power of spiking neural P systems, in: M.A. Gutiérrez-Naranjo et al. (Eds.), Proceedings of the Fifth Brainstorming Week on Membrane Computing, Sevilla, Jan. 2007, 227–245.

[7] LEPORATI, A., ZANDRON, C., FERRETTI, C., MAURI, G., Solving numerical NP-complete problems with spiking neural P systems, in: George Eleftherakis et al. (Eds.), Proceedings of the Eightth Workshop on Membrane Computing, Thessaloniki, June 2007, 405–423.

[8] MINSKY, M. L., Computation. Finite and Infinite Machines, Prentice-Hall, 1967.

[9] NEARY, T., On the computational complexity of spiking neural P systems, in: 7th International Conference on Unconventional Computation (UC 2008), Vienna, Aug. 2008, Lecture Notes in Computer Science 5204, Springer, 2008, 190–20.

[10] PĂUN, A., PĂUN, Gh., Small universal spiking neural P systems, BioSystems 90 (1) (2007), 48–60.

[11] ZHANG, X., ZENG, X., PAN, L., Smaller universal spiking neural P systems, submitted, 2008.

AN EXTENDED DOT-BRACKET-NOTATION FOR FUNCTIONAL NUCLEIC ACIDS

Effirul I. Ramlan and Klaus-Peter Zauner
School of Electronics and Computer Science
University of Southampton, SO17 1BJ, United Kingdom
Email: {eir05r,kpz}@ecs.soton.ac.uk

Abstract
Functional nucleic acids are an attractive substrate for molecular computing. A nucleic acid molecule is a linear chain of covalently bound building blocks assembled in arbitrary order from a set of typically four nucleotides. Certain pairs of nucleotides weakly attract each other through short-range electrostatic interaction and, accordingly, complementary sequences of nucleotides can bind to each other. The complementary stretches of nucleic acids that attract each other can be part of two different molecules or two parts of a single molecule. Binding within a single molecule leads to a folding of the linear chain. This so called secondary structure is of great importance for the function of nucleic acids.

The present paper is concerned with the representation of this secondary structure. We propose an extension for the syntax of the standard dot-bracket notation to increase its convenience and expressive power for both its use to communicate nucleic acid secondary structures among humans and machines. The extensions reflect our own requirements for the representation of nucleic acids for molecular computation, but should be useful for functional nucleic acids in general.

1. Computational Nucleic Acid Enzymes

Organisms have powerful and enviably efficient information processing capabilities. To a large extent these capabilities are conferred by macromolecules and their specific properties. The existence of these natural information processing architectures demonstrates that computing based on physical substrates that are radically different from silicon is feasible. Accordingly, the potential of biomolecules as a computing substrate in artificial devices has been investigated for over three decades [15]. In nature proteins appear to play the preeminent role as molecular computing substrate. At the present state of technology, however, two other classes of biomolecules are more amenable to applications in man-made information processing architectures: deoxyribonucleic acids (DNA) and ribonucleic acids (RNA). The former offers a more limited conformational flexibility and concomitant less functionality, while the latter is less stable and requires more careful laboratory techniques. The existing body of work on using nucleic acids for information processing can be grouped under three concepts:

Covalent Concept: Early proposals for the use of DNA in computing were inspired by the discovery of DNA's role in the storage of inheritable information and the astounding information density that can be achieved with molecular encoded data. All of these concepts require the formation and cleavage of specific covalent bonds which would require custom-designed proteins and are for this reason not practical (cf., e.g., [8])

Complement Concept: In [1] the influential suggestion to use arbitrary nucleotide sequences and the hybridisation with their complementary sequences instead of covalently linked individual bases was made and a practical demonstration of this approach was given. This idea moved the burden of recognising tokens of information from proteins (which up to now cannot be designed for purpose) to the self-assembly of short nucleotide sequences which can be designed with desired self-assembly properties and can be synthesised with ease.

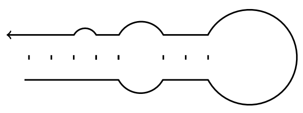
Conformational Concept: Whereas in the above two concepts the conformational flexibility of the nucleic acids is irrelevant or even undesirable, more recently computing concepts that exploit the change in conformation a nucleic acid undergoes upon hybridising with another nucleic acid molecule have been developed [9, 10].

A recent introduction to the conformational concept of information processing with nucleic acids is available in [11]. The three-dimensional conformation of a nucleic acid is largely determined by the secondary structure, i.e., the intramolecular binding among complementary sections of its nucleotide sequence. The sequence itself is typically for the most part not as important for the function as the secondary structure it assumes. For a given secondary structure there is generally a large variety of nucleotide sequences that it will fold into.

In designing sets of nucleic acids for information processing one typically has a desired secondary structure and additional local constraints on the sequence. For example, a binding side for an effector molecule (i.e. a nucleic acid that will affect the activity of a functional nucleic acid) may be required to be complementary to a sequence released in a preceding step. Software tools that, given a secondary structure, can generate a suitable sequence candidate likely to fold into the desired structure are available [2, 3, 5]. It would be desirable to have a convenient representation of secondary structures together with sequence constraints and special properties of regions of the sequence. Ideally a single representation should support the communication between humans and software.

2. Representation of Nucleic Acid Structure

A widely used method to denote RNA secondary structure is the dot-bracket-notation or parenthesis format introduced by Hofacker et al. [2]. It uses matching parenthesis and dots to denote paired and free bases, respectively. Fig. 1 illustrates the notation for a short RNA sequence.

A 

B `CCGAUAGAGGCGUGCGGUCAAGGUCCGG`
C `(((((...(((.....)))...)).)))`

Figure 1: A sample secondary structure of an RNA molecule is shown (A) together with the notation of its sequence (B) and its structure in dot-bracket form (C).

The dot-bracket-notation has the advantage that a string denoting the secondary structure of a nucleic acid is of the same length as the string denoting the nucleotide sequence with a single character for each nucleotide. The two strings can be aligned to show the secondary structure features along the nucleotide sequence (Fig. 1B and C). In molecular biology one typically has a given (discovered) sequence and is interested in its folding properties. The dot-bracket notation reflects this mode of operation. As indicated above, in molecular computing applications it is common that the secondary structure is of importance, but the detailed sequence that yields the structure is over large stretches arbitrary. In such a scenario the dot-bracket notation is often cumbersome, it leads to large expressions with information that is partially obscured for human readers, because it would require counting identical characters. While the sequence is typically arbitrary in most positions, as long as the structure of the molecule is preserved, nucleic acids with functional properties, such as catalytic activity, often require specific bases in a few positions. If in a few places the nucleotide sequence (i.e. the primary structure) is given, two strings are required: one to specify the structure in dot-bracket-notation and a second string to represent the type of the immutable nucleotides. Furthermore, for communicating structural features among humans a two-dimensional rendering of the one-dimensional dot-bracket notation is often desirable. It would be convenient if specific features in the sequence could be communicated to the rendering software.

To alleviate these issues we use an extended dot-bracket-notation that is particularly suitable to denote functional nucleic acids where the primary characteristic is the secondary structure and not the sequence. In doing so we were aiming at:

- backward compatibility to the dot-bracket notation
- use of familiar and mnemonic conventions
- single character operators
- the possibility to describe sets of interacting molecules
- flexibility to chose expressions according to application
- support for rendering with and without colour

Achieving these aims comes at the price of giving up the equivalence of the length between the secondary structure specification and the sequence. On the other hand, the extended notation is often more compact and capable of describing, in a single string, a group of RNA molecules where each molecule varies in length, sequence, and conformation.

The extended dot-bracket-notation introduces several new symbols which fall into four different categories:

Scoping symbols group sections of the notations. Square brackets "[ ]" are used for grouping range of base positions. Curly braces delimit alphanumerical parameters for operators and also limit sets of constraints placed on the choice of base permitted for a particular sequence position. Curly braces can optionally be used to delimit the numbers specifying repetitions.

Operator symbols associate a property with the preceding base position, or grouped range of positions. The properties are mostly used for graphical rendering of the the structure (`_`,`$`,`~`,`@`), but also to mark binding sites for inter-molecular binding (`+`). An operator symbol is always followed by a parameter.

Constraint symbol restrict the possible bases that may be present at the preceding base position. The two constraint symbols are a colon (`:`) which restricts the preceding position to be equal to the base or set of bases that follow it, and a hat (`^`) that restricts the preceding position to differ from the base or set of bases that follow it.

Special symbols are available to express features which cannot be expressed with the above elements. At present only the `%`-symbol is defined, it marks a cleavage-point where the RNA sequence may be hydrolysed.

An overview of the new symbols introduced in the extended dot-bracket-notation is provided in Tab. 1. With these enhancements the extended dot-bracket notation can carry a lot more information about an RNA structure than the standard notation while generally leading to a more compact description. However, the translation from the extended dot-bracket-notation to the standard notation is trivial. This is important as the computational tools for nucleic acids secondary structure have adopted the dot-bracket-notation as their input–output channels [6]. A translation from the standard notation to the extended dot-bracket-notation can of course not make much use of the richer syntax of the extended notation. It is also generally not required, as the standard notation is a valid subset of the extended notation. Nevertheless, such a translation may be useful to arrive at shorter representations as shown for small examples in Tab. 2. Any symbol that can occupy a base position in a sequence (i.e., `.`,`(`,`)`,`A`,`U`,`G`,`C`,`T`,`...`) may be followed by a positive integer value n to denote n repetitions of the symbol. This run-length notation is particularly convenient for manually entering secondary structure descriptions. The downside is that the run-length notation can obscure structural motives which may be recognised more readily in the standard notation. A considered use of the repetition parameter will maximise readability, whether by reducing the length of the representation or by deliberately breaking runs of parenthesis into sections that match. For example, `(3.2(3.4)6` represents a stem-loop with bulge. The same structure could also be written as `(3.2(3.4)3)3`.

Table 1: New symbols introduced in the extended dot-bracket-notation

	Description	Usage	Comment
`[ ]`	Grouping of base positions	`[.8]@{label A}`	Eight unbound bases marked as “label A”.
`{ }`	Parameter delimiter		see example above
`{ }`	Set delimiter	`.:{A,C}`	A single unbound base that can be either A or C.
`{ }`	Repetition delimiter	`A{10}`	Always optional.
`_`	Line width	`((([.5]_1)))`	Stem-loop structure with bold loop
`$`	Colour	`(3[.2]$1(3.4)3)3`	A buldge in red.
`~`	Line decoration	`.24~1(3.3)3`	Binding site marked as crinkled line.
`@`	Annotation marker		See first row.
`+`	Multi-molecule binding	`(24+1(3.3)3`	Sticky end of 24 bases, will bind to site marked 1 on other molecule.
`:`	Base assignment	`)):A`	Two binding bases, the second one of which is A; See also set delimiter.
`^`	Base exclusion	`(((..^U.)))`	Stem loop where the central base in the loop is not a uracil.
`%`	Clevage point	`(((..%..(((`	Between bases, i.e., not a base position.

The latter is longer, but preferable nevertheless, because the base-pairing of the two helices is emphasised by breaking the run of six closing parenthesis into two groups of three closing parenthesis each. This example also illustrates that in the extended dot-bracket-notation there is no unique string to describe a given structure. The equivalence of two structures denoted in the extended form, however, can be established by translating both into the standard form, which is easily accomplished.

Specifications for individual base positions, repetitions of these, as well as groups (marked by square brackets) of individual positions and repetitions can be arguments for operators. The operator follows its argument and precedes its parameters. More than one operator/parameter combination may follow an argument and all will be applied to the argument. Table 3 provides a few examples of operator use—some of them taken from the structures rendered in Figs. 3

Table 2: The extended dot-bracket-notation allows for run-length encoding to achieve a compact representation

Standard notation	Extended notation	Part of Fig.
`...(((((....))))).))))))))`	`.3(5.4)5.)8`	3A
`(((....))).....((((((((`	`(3.4)3.5(8`	3B
`(((((....))))).)))))))).`	`(5.4)5.)8.`	4A
`..)).)))........`	`.2)2.)3.8`	4C

Table 3: Sample usage of operators

Notation	Description	Fig.
`~2@{shift region 1}`	Apply decoration type 2 to the preceding region and label it as "shift region 1".	3A
`+3~2`	The preceding argument binds externally with another molecule at the region marked "3" and is drawn with decoration type 2 ("cross").	3C
`+2_1@{OBS2+EFF2}`	The argument (not shown) binds with another molecule at the region marked "2", draw the binding region in bold (line thickness 1) and label it as "OBS2+EFF2"	4C
`+1_1${Red}@{node001}`	The preceding argument binds externally with another molecule at the region marked "1", render the argument with line thickness 1 in red and label the region as "node001"	-
`~1_2${blue}`	Combination of drawing parameters applied to the preceding argument resulting in a strong bold (thickness 2) crinkled line (type 1) in blue color.	-
`+{SITE1}~1_1@{match}${red}`	The preceding argument binds externally with another molecule at the region marked as "SITE1". This region is rendered using crinkle line with the thickness value 1, and colored in red. The region is labeled "match".	-

and 4. Note that in all cases the argument itself, which precedes the operator, is not shown.

The acceptable parameters that follow an operator and their semantics are not specified by the notation. A rendering program, for example, may accept a predefined color number, an explicit color name, or a hexadecimal RGB value. The corresponding operator with parameters would be `$1`, `${red}`, and `${#FF0000}`. An overview of the extended notation is provided in Fig. 2. For clarity, three of the nonterminal symbols occurring in the syntax graph are not shown in Fig. 2. The non-terminal *digit* stands for a single digit in the range from 0–9. The non-terminal *alpha* stands for a single character from either the range a–z, or A–Z, or a dash (-), underline (_), or space ( ). The non-terminal *base* stands for any one of (A,U,G,C,T,X,N) in upper or lower case.

As can be seen in Fig. 2, a single string in the extended dot-bracket-notation can denote more than one molecule (cf. *RNA_string* in Fig. 2). The limitations in expressing interactions among multiple molecules in the standard notation was one of the factors motivating the present work. An example with a pair of molecules that bind in two different regions of equal length will illustrate the difficulty of using the standard notation in such cases:

$$\begin{array}{l}\underline{\texttt{((((((}}\texttt{.....((((((......))))))...((((....))))...}\underline{\texttt{((((((}}\texttt{ \&}\\ \underline{\texttt{))))))}}\texttt{........(((((....((((....))))....)))))...}\underline{\texttt{))))))}}\end{array}$$

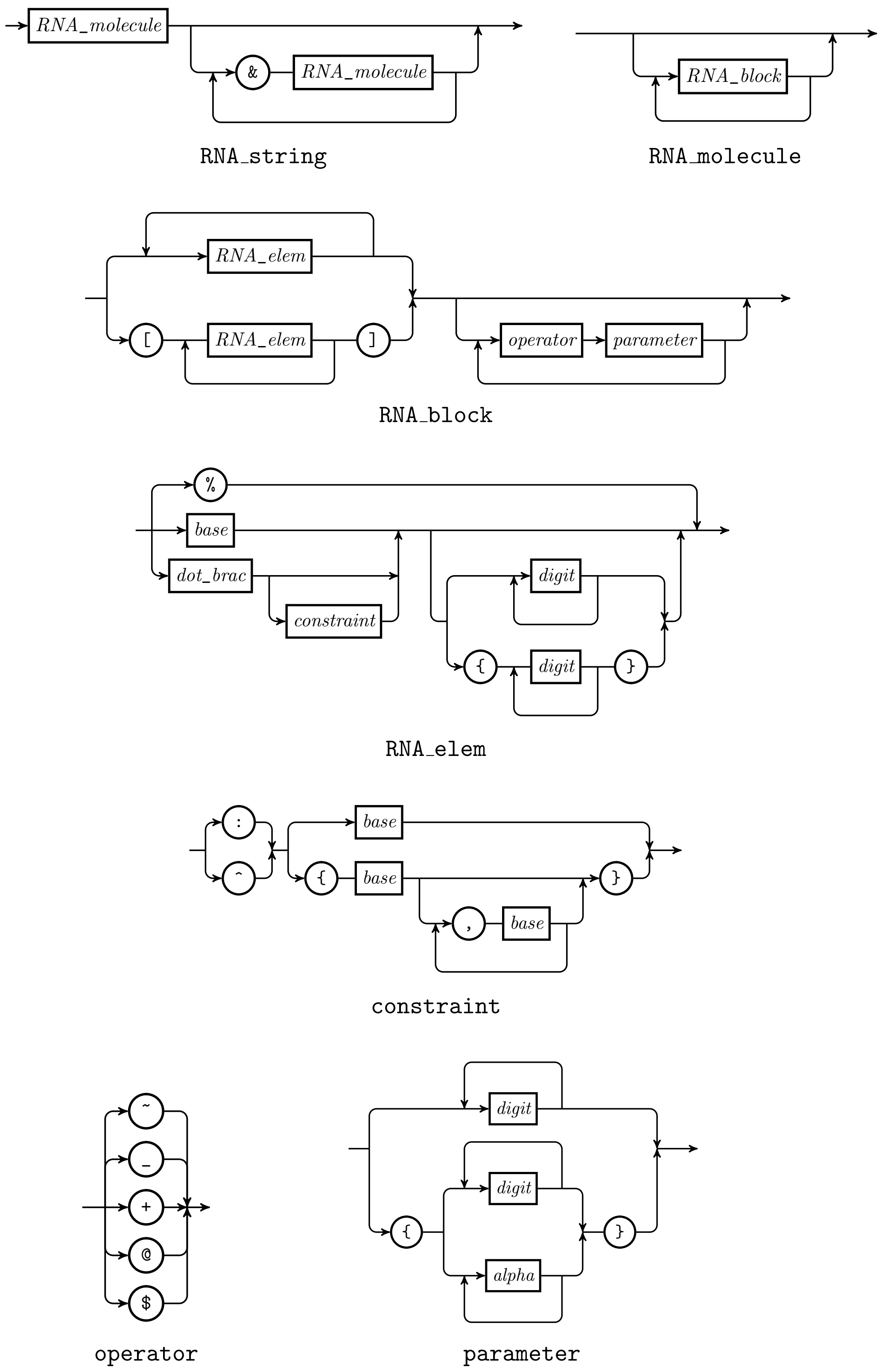


Figure 2: Syntax graph for the extended dot-bracket notation.

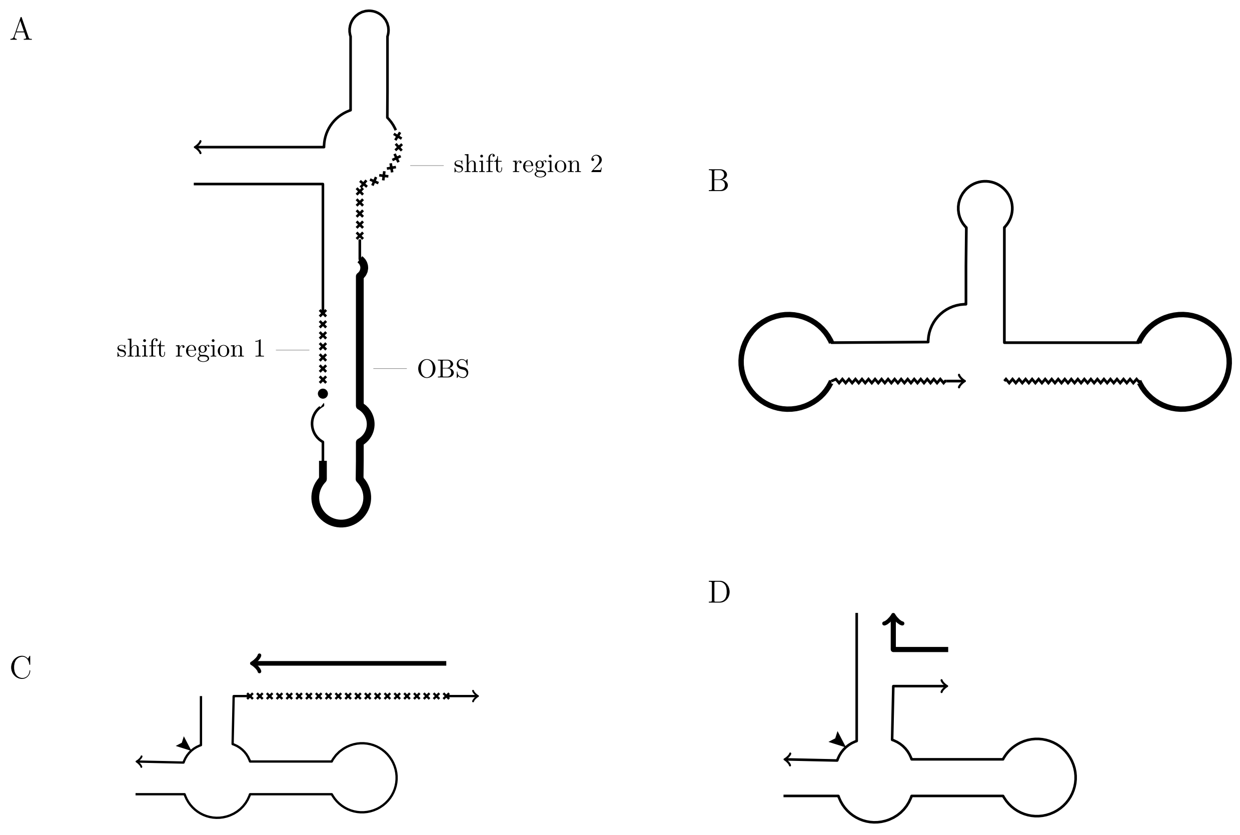
The two lines represent two different molecules, separated by the &-symbol. The regions in which the two molecules will bind to each other are underlined. The dot-bracket-notation is not able to express in which combination the binding regions will bind. In larger molecules the situation can easily be more ambiguous with numerous plausible locations for intermolecular binding. In the extended dot-bracket-notation, the + operator can indicate the matching regions. Accordingly, the two molecules shown above can be represented as:

```
(6+{B1}.....((((((......))))))...((((....))))...(6+{B2} &
)6+{B1}........(((((....((((....))))....)))))...)6+{B2}
```

A more relevant example for such an ambiguous binding situation can be seen in Fig. 4D, where the two binding sites in the AND gate have the same length. The AND gate uses two effector molecules as input signals and, in its active state, is a three-molecule supramolecular complex. The alphanumeric marking of binding sites in the extended dot-bracket-notation enables the description of interactions among several molecules. Note in the examples above, how the standard notation and the extended notation can be mixed to highlight particular features of a molecule or set of molecules.

The benefit of the extended notation is most easily seen when it is rendered as two-dimensional structures. Figure 3 shows the rendering of several sample structures from the literature. Panel A shows a ribonucleic acid PASS gate in its inactive state [10]. Panel B depicts a deoxyribonucleic acid AND gate described in [12]. The mechanisms of Fig. 3A is based on disrupting the active conformation of a nucleic acids enzyme. Upon binding of an effector molecule to the binding site shown in bold the active conformation is restored. In the gate depicted in Fig. 3B, the binding site for the substrate of a nucleic acids enzyme is blocked by intramolecular binding highlighted with a crinkled lines. Binding of effector molecules to the two binding sites shown in bold will expose the substrate binding site. Two hammerhead ribozymes with different catalytic activation strategies described in [4] and [14] are shown in Fig. 3C and D, respectively. The hammerhead ribozymes are shown in their active conformation, each with a bound effector molecule. The effector molecules are marked with bold lines and the location of the cleavage point is marked on the substrate strand. The corresponding extended notation for the four structures is shown in panel E.

In Fig. 4 four different states of the ribonucleic AND gate designed by Penchovsky and Breaker [10] are shown. The interplay of multiple molecules and multiple conformational states is crucial to the computing schemes based on functional nucleic acids. They are also a challenge to represent in a convenient notation. Panels A shows the secondary structures of the AND gate without effector molecules, panels B and C show the structures the AND gate assumes if only one of the effector molecules is present. If both oligonucleotide binding sites (OBS1, OBS2) are occupied by effector molecules the ribozyme changes into the catalytically active conformation shown in panel D. The extended dot-bracket-notation corresponding to the four states of the gate are shown in panel E.

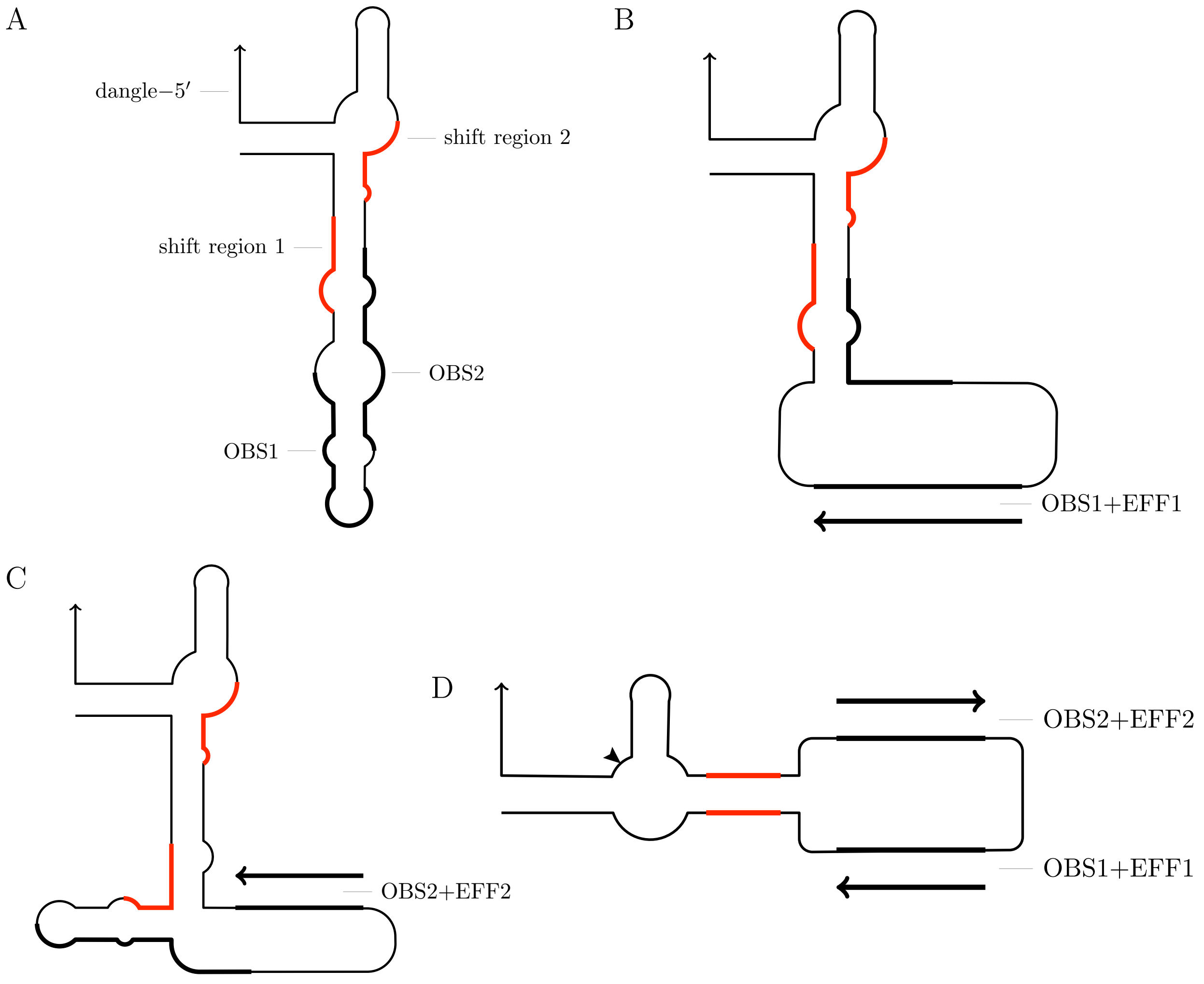


E

A	`({15}[(3]~2@{shift region 1}[(3]_2(.4[(2.7)3.)2]_2@{OBS}[)3.]_2`
	`[(4.2]~2@{shift region 2}.3(5.4)5.)8`

B	`[(9]~1[.{15}]_1)9(3.4)3.5(8[.15]_1[)8]~1`
C	`[(5]+1.7(4.4)4.3[(5]+2.2[(8]+3~2.2 & [)5]+2[.]%[(5]+1 & [)8]+3_2`
D	`[(8]+1.7(4.4)4.3[(3]+2[(6]+3 & [)6]+3_1[(6]+4_1 & [)6]+4[)6]+2[.]%[)8]+1`

Figure 3: Four different structural renderings of arbitrary nucleic acid molecules.



E

A	`({15}[(4]$1@{shift region 1}[(.4]$1(3.3[.2(3.]_1@{OBS1}[(3.7]_1 )3[.)3.3]_1@{OBS2}[.3)3.2)2]_1)6[.)5]$1[.3]$1@{shift region 2}.3 (5.4)5.)8.9`
B	`({15}[(5.3]$1(3.3[({16}]+1_1.3[.9)3.2)2]_1)5[.)5.2]$1.3(5.4)5.)8.9 & [){16}]+1_1@{OBS1+EFF1}`
C	`({15}[(3.)3.]$1.2(2.3[.2)2.)3.8]_1.3[({16}]+2_1)5[.)5.2]$1.3(5.4)5.)8.9 & [){16}]+2_1@{OBS2+EFF2}`
D	`(8.7[(8]$1.6[({16}]+1_1.3[({16}]+2_1.5[)8]$1.3(5.4)5[.]%)8.9 & [){16}]+1_1@{OBS1+EFF1} & [){16}]+2_1@{OBS2+EFF2}`

Figure 4: RNA molecular AND gate after [10] in different states. Rendered from the extended dot-bracket notation.

3. Conclusions

Within recent years nucleic acids of up to about 200 nucleotides in length have become a focus of interest for prototype implementations of molecular computing concepts. During the same period the importance of ribonucleic acids as components of the regulatory networks within living cells has increasingly been revealed. While the configuration of the nucleic acids is generally linear, they can adopt a range of conformations. The conformation adopted by a nucleic acid is determined by the possibility of its nucleotide chain to form non-covalent intramolecular bonds through hybridisation of complementary regions in the nucleotide sequence. This intramolecular hybridisation pattern, known as secondary structure of the molecule, is crucial to the interactions a nucleic acid undergoes with other molecules. For a given secondary structure there is typically a large number of sequences that will adopt this structure. The standard method for denoting nucleic acid secondary structure is in the form of a string of matching parenthesis for hybridising pairs of nucleotides separated by dots representing nucleotides that do not participate in internal hybridisation. We have extended this notation for the convenience of human users as well as machine processing. The extensions allow for a more compact notation through the use of iterator operators and grouping symbols, provide for constraints placed on the nucleotides that may appear in a position, and facilitate the annotation of sequence regions and the graphical rendering of secondary structures.

A downside of the proposed annotation is the potentially increased complexity of the strings that represent a molecule. For human readers this means that more symbols and the scoping of operators needs to be understood to read the notation. For machines establishing the equivalence of the secondary structures denoted by two strings is no longer as simple as comparing the strings. However, the language of the dot-bracket notation is a subset of the extended dot-bracket notation proposed here and the use of its symbols and operators is thus wholly optional. The most appropriate notation will depend on the application. For instance, the use of iterator operators make it easier for humans to compare the length of hybridised regions within a molecule, but the relative lengths of oligonucleotides is more readily apparent if denoted without iterator operators. Human users can choose to use the features of the extended notation according to application. For machine processing the extended notation can easily be expanded or reduced to the standard dot-bracket notation.

In our own work we felt the need for a more expressive notation for computational nucleic acids. The extended dot-bracket notation described here answers this need. We are currently converting our software tools to the new notation. The accompanying widening of its use is likely to lead to further refinements of the notation and we welcome suggestions for improving its usefulness. For example, the application of iterators to groups and the nesting of groups are not currently part of the extended notation, but could be added if needed. We expect to make the code used for rendering secondary structures in LaTeX [7] with TikZ [13] available in the near future.

References

[1] ADLEMAN, L. M., Molecular computation of solutions to combinatorial problems, Science 226 (1994), 1021–1024.

[2] HOFACKER, I. L., FONTANA, W., STADLER, P. F., BONHOEFFER, L. S., TACKER, M., SCHUSTER, P., Fast folding and comparison of RNA secondary structures, Chemical Monthly 125 (2) (1994), 167–188.

[3] ANDRONESCU, M., ANGUIRRE-HERNÁNDEZ, R., CONDON, A., HOOS, H. H., RNAsoft: A suite of RNA secondary structure prediction and design software tools, Nucleic Acid Research 31 (13) (2003), 3461–3422.

[4] BURKE, D. H., OZEROVA, N. D. S., NILSEN-HAMILTON, M., Allosteric hammerhead ribozyme TRAPs, Biochemistry 41 (2002), 6588–6594.

[5] BUSCH, A., BACKOFEN, R., INFO-RNA – a fast approach to inverse RNA folding, Bioinformatics 22 (15) (2006), 1823–1831.

[6] HIGGS, P. G., RNA secondary structure: physical and computational aspects, Quarterly Reviews of Biophysics 33 (3) (2000), 199–253.

[7] LAMPORT, L., LaTex: A document preparation system, user's guide and reference manual, Addison-Wesley, Mass 1994.

[8] LIBERMAN, E. A., Analog-digital molecular cell computer, BioSystems 11 (1979), 111–124.

[9] STOJANOVIC, M. N., STEFANOVIC, D., A deoxyribozyme-based molecular automaton, Nature Biotechnology 21 (9) (2003), 1069–1074.

[10] PENCHOVSKY, R., BREAKER, R. R., Computational design and experimental validation of oligonucleotide-sensing allosteric ribozymes, Nature Biotechnology 23 (11) (2005), 1424–1433.

[11] RAMLAN, E. I., ZAUNER, K.-P., Nucleic acid enzymes: The fusion of self-assembly and conformation computing, International Journal of Unconventional Computing, in print (2008).

[12] STOJANOVIC, M. N., MITCHELL, T. E., STEFANOVIC, D., Deoxyribozyme-based logic gates, Journal of the American Chemical Society 124 (2002), 3555–3561.

[13] TANTAU, T., TikZ and PGF Manual for version 2.0, 2007.

[14] WANG, D. Y., LAI, H. Y., FELDMAN, A. R., SEN, D., A general approach for the use of oligonucleotide effectors to regulate the catalysis of RNA-cleaving ribozymes and DNAzymes, Nucleic Acids Research 30 (8) (2002), 1735–1742.

[15] ZAUNER, K.-P., Molecular information technology, Critical Reviews in Solid State and Material Sciences 30 (1) (2005), 33–69.

A SOFTWARE TOOL FOR GENERATING GRAPHICS BY MEANS OF P SYSTEMS

Elena Rivero-Gil Miguel A. Gutiérrez-Naranjo
Álvaro Romero-Jiménez Agustín Riscos-Núñez
Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville
E-mails: elen.rg@gmail.com, magutier@us.es,
Alvaro.Romero@cs.us.es, ariscosn@us.es

Abstract
The hand-made graphical representation of the configuration of a P system becomes a hard task when the number of membranes and objects increases. In this paper we present a new software tool, called JPLANT, for computing and representing the evolution of a P system model with membrane creation. We also present some experiments performed with JPLANT and point out new lines for the research in computer graphics with membrane systems.

1. Introduction

Since A.R. Smith [12] proposed Lindenmayer systems (L-systems) [5] as a tool for synthesizing realistic images of plants, many efforts have been done for bridging the theory of formal languages and computer graphics.

In [2, 3], a first membrane-based device for computer graphics was presented. It was a hybrid model between L-systems and membrane computing and it used concepts very close to the L-systems model. Later, in [10], a new approach was presented for representing the development of higher plants with P systems. It was based on a type of P systems with membrane creation and it was entirely developed with membrane computing techniques. The basic idea was to consider the growing of the structure of membranes in a P system with membrane creation.

By definition, the structure of membranes in a cell-like P system is a (formal) tree. In P systems with membrane creation, new membranes can be created inside the existing membranes and this produces the expansion of the structure of membranes by increasing the depth of the branches. With an appropriate interpretation of the objects inside the membranes, the membrane structure can be represented as a tree which evolves in time and the length and width of the branches can grow in a similar way to real plants. In [11], the study started at [10] was completed by adding stochastic rules to the P system. In this case, the random choice of different rules produces different configurations of the P system and, hence, different graphical representations.

The hand-made graphical representation of the configurations of the P system becomes a hard task when the number of membranes and objects increases. For this reason, to study in depth the relationship between P systems and computer graphics it was necessary to develop a software able to deal with complex P systems and to represent graphically its evolution along time.

In this paper we present such a software, JPLANT, which computes the first configurations of a computation and draws the corresponding graphical representations. This software is a very useful tool for the experimental research of the graphical representation of P systems. We show several experiments and open new research lines for exploring the possibilities of P systems.

The paper is organized as follows: Section 2 recall the restricted model of P systems with membrane creation used for the graphical design. Section 3 gives a brief presentation of the software JPLANT and the next section shows several experiments. The paper finishes with some conclusions and lines for future research.

2. P Systems with Membrane Creation

Membrane computing is a branch of natural computing which abstracts from the structure and the functioning of the living cell. In the basic model, membrane systems (also frequently called P systems) are distributed parallel computing devices, processing multisets of symbol-objects, synchronously, in the compartments defined by a cell-like membrane structure (a detailed description of P systems can be found in [9] and updated information in [13]).

In this paper we consider P systems which make use of membrane creation rules, which were first introduced in [4, 6]. However, our needs are far simpler than what the models found in the literature provide. This is the reason why we introduce the new variant of *restricted P systems with membrane creation.*

A restricted P system with membrane creation is a tuple $\Pi = (O, \mu, w_1, \dots, w_m, R)$ where:

1. O is the alphabet of *objects*. There exist two distinguished objects, F and W, that always belong to the alphabet of any P system considered below.

2. μ is the initial *membrane structure*, consisting of a hierarchical structure of m membranes (all of them with the same label; for the sake of simplicity we omit the label).

3. $w_1, \dots, w_m$ are the multisets of objects initially placed in the m regions delimited by the membranes of μ.

4. R is a finite set of *evolution rules* associated with every membrane, which can be of the two following kinds:

(a) $a \to v$, where $a \in O$ and v is a multiset over O. This rule replaces an object a present in a membrane of μ by the multiset of objects v.

(b) $a \to [v]$, where $a \in O$ and v is a multiset over O. This rule replaces an object a present in a membrane of μ by a new membrane with the same label and containing the multiset of objects v.

A membrane structure together with the objects contained in the regions defined by its membranes constitute a configuration of the system. A transition step is performed applying to a configuration the evolution rules of the system in the usual way within the framework of membrane computing, that is, in a non-deterministic maximally parallel way; a rule in a region is applied if and only if the object occurring in its left–hand side is available in that region; this object is then consumed and the objects indicated in the right–hand side of the rule are created inside the membrane. The rules are applied in all the membranes simultaneously, and all the objects in them that can trigger a rule must do it. When there are several possibilities to choose the evolution rules to apply, non-determinism takes place.

2.1. Graphical Representation

In this section we show how to use, through a suitable graphical representation, restricted P systems with membrane creation to model branching structures. The key point of the representation relies on the fact that a membrane structure is a *rooted tree of membranes*, whose root is the skin membrane and whose leaves are the elementary membranes. Thus, this seems a suitable frame to encode the branching structure.

Let us suppose that the alphabet O of objects contains the objects F and W, and let us fix the lengths l and w.

A simple model to graphically represent a membrane structure is to make a depth-first search of it, drawing, for each membrane containing the object F, a segment of length $m \times l$, where m is the multiplicity of F. If the number of copies of F in a membrane increases along the computation, the graphical interpretation is that the corresponding segment is lengthening. Analogously, the multiplicity of the symbol W specifies the width of the segments to be drawn as follows: if the number of objects W present in a membrane is n, then the segment corresponding to this membrane must be drawn with width $n \times w$.

Each segment is drawn rotated with respect to the segment corresponding to its parent membrane. In order to determine the rotation angle we need to fix a third parameter δ. This angle δ together with the length l and the width w will determine the picture of each configuration of the P system.

In order to compute the rotation angle of a segment with respect to the segment corresponding to its parent membrane we consider two new objects that can appear in the alphabet: $+$ and $-$. The rotation angle will be $n \times \delta$, where n is the multiplicity of objects "+" minus the multiplicity of objects "$-$" in the membrane. That is, each object "$+$" means that the rotation angle is increased by δ whereas each object "$-$" means that it is decreased by δ.

Inside the membranes other objects can appear that do not have geometrical interpretation. They are related to the development of the graph along time.

For a better understanding let us consider the following example: let Π_1 be the restricted P system with membrane creation such that

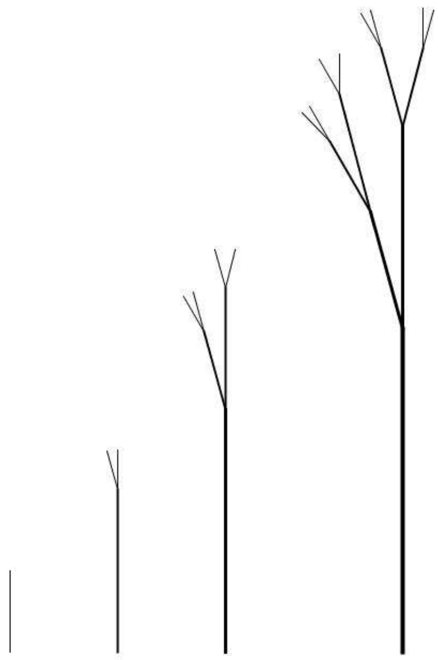


Figure 1: First four configurations

- The alphabet of objects is $O = \{F, W, B_l, B_s, B_r, L, L_1, E, +, -\}$.
- The initial membrane structure together with the initial multiset of objects is $[F^2\, W\, B_l\, B_s\, L_1\, E]$.
- The rules are:

$$\begin{array}{llll} B_l & \rightarrow [+\, F\, W\, B_l\, B_s\, L\, E] \qquad & L & \rightarrow L\, F \\ B_s & \rightarrow [F\, W\, B_l\, B_r\, L_1\, E] & L_1 & \rightarrow L_1\, F^2 \\ B_r & \rightarrow [-\, F\, W\, B_l\, B_s\, L\, E] & E & \rightarrow E\, W \end{array}$$

In this system, the object B_s represents the straight branches to be created, whereas the objects B_l and B_r represent branches to be created rotated to the left and to the right, respectively. The objects F and W will determine the length and the width of the corresponding branch. The objects L, L_1, and E do not have a graphical interpretation; they can be considered as seeds for growing the branch in length and width.

The initial configuration consists of one membrane which contains two copies of F and one copy of W. If we consider the parameters l, w, and δ, then the graphical representation of this initial configuration is a single segment of length $2 \times l$ and width w. In the first step, the objects B_l and B_s create new membranes, so the picture of this configuration consists on three segments. The new membrane created by B_s does not contains objects $+$ or $-$ and then the corresponding segment is not rotated with respect to the segment that represents the skin. On the other hand, the membrane created by B_l contains one object $+$, so its segment will be rotated an angle δ with respect to its parent membrane.

Notice also that the evolution of the objects L_1 and E has modified the number of objects F and W in the skin, so in this new picture, the segment corresponding to the skin has length $4 \times l$ and width $2 \times w$.

Figure 1 shows the graphical representation of the first four configurations where we fix a bottom-up orientation and an angle δ of 15 degrees.

2.2. Stochastic Versus Non-deterministic P Systems

The non-determinism is one of the main features of P systems and the possibility of reaching different configurations leads us to consider different graphical representations in the evolution of a P system.

One possible way to formalize the probability of obtaining one or another configuration is via stochastic P systems. Several alternatives to incorporate randomness into membrane systems can be found in the literature (see [1, 7, 8] and the references therein). One of them is to associate each rule of the P system with a probability of execution. Thus, to pass from a configuration of the system to the next one we apply to every object present in the configuration a rule chosen at random, according to those probabilities, among all the rules whose left–hand side coincides with the object (this idea was presented in [11]).

For example, let us consider Π_2 the following restricted P system with membrane creation:

- The alphabet of objects is $O = \{F, W, B_l, B_s, B_r, L, L_1, E\}$.
- The initial membrane structure together with the initial multiset of objects is $[F^2\, W\, B_l\, B_s\, L_1\, E]$.
- The rules are:

$$\begin{array}{llll}
B_l & \xrightarrow{1/2} [+\,F\,W\,B_l\,B_s\,L\,E] & \quad L & \rightarrow L\,F \\
B_l & \xrightarrow{1/2} [-\,F\,W\,B_l\,B_s\,L\,E] & \quad L_1 & \rightarrow L_1\,F^2 \\
B_r & \xrightarrow{1/2} [+\,F\,W\,B_l\,B_s\,L\,E] & \quad E & \rightarrow E\,W \\
B_r & \xrightarrow{1/2} [-\,F\,W\,B_l\,B_s\,L\,E] & \quad B_s & \rightarrow [F\,W\,B_l\,B_r\,L_1\,E]
\end{array}$$

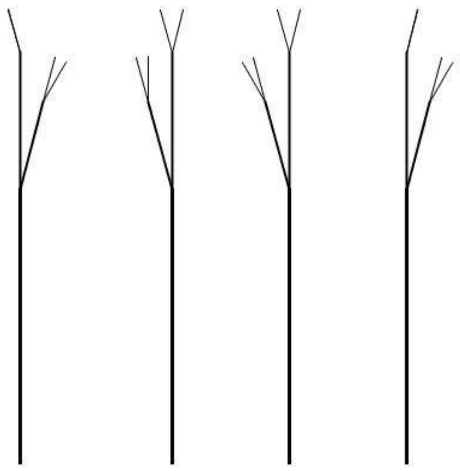


Figure 2: Four configurations after the second step

There exist two rules for the evolution of the object B_l and two possibilities for the evolution of the object B_r. The probability for each choice is 1/2. Notice that we do not make explicit the probability of the rule when this is 1.

Figure 2 shows four different configurations after the second step of this P system with the angle $\delta = 15$.

3. JPLANT

In order to avoid the heavy task of obtaining by hand the graphical representation of a configuration of a restricted P system with membrane creation a new software tool has been designed. In this paper we present JPLANT (the software is available from [13]), which computes the first configurations of a computation of such a P system and draws the corresponding graphical representations of these configurations.

JPLANT has been written in Java and it has an intuitive user-friendly graphical interface. The initial configuration and the set of rules are provided in plain text mode. The syntax of the initial configuration and the rules of the system are checked for correctness before starting the computation. The generation of a new configuration is driven by the user which can choose between jumping to a configuration N or generating (and drawing) at each time the next configuration.

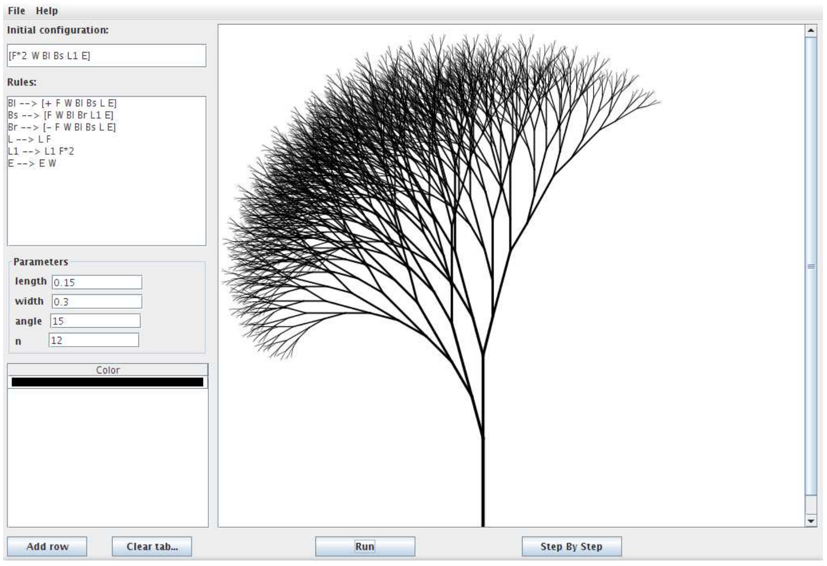
The software tool is thought as a drawing tool so the computed new configurations are not showed to the user in text mode. The output is a picture with a set of connected segments drawn according with the rules described in Section 2. For each new configuration, a new picture is drawn, so the output of this tool is a sequence of pictures which can be saved in several computer graphic formats.

The graphical representation of one configuration is not unique. It depends on the parameters l, w and δ which determine the length and width of the segments as well as the angles between them. Such parameters must be also provided by the user as input of the tool, together with the initial configuration and the rules.

The current version of JPLANT includes the ability to load and save files with the input data and to save the generated pictures and also to provide color to the pictures.

Another feature of JPLANT is the possibility of coloring the graphics generated from the configurations of the system. This is performed bu drawing the segments associated with each membrane in different colors, obtaining this way more realistic pictures. We must point out that these graphical elements are not intrinsic to the P system, but are indicated externally to it.

Figure 3 shows a snapshot of this computer software.



Initial configuration:

```
[F*2 W Bl Bs L1 E]
```

Rules:

```
Bl --> [+ F W Bl Bs L E]
Bs --> [F W Bl Br L1 E]
Br --> [- F W Bl Bs L E]
L --> L F
L1 --> L1 F*2
E --> E W
```

Parameters

length	0.15
width	0.3
angle	15
n	12

Color
■

Figure 3: Snapshot of JPLANT

4. Applications

Next we illustrate the possibilities of JPLANT with some examples.

4.1. Polygons and Spirals

Polygons and spirals can be considered a very special case of branching structures. They consist of a connected set of segments where a vertex only connect two segments. From a membrane computing point of view, this means that each membrane in a configuration only contains one membrane.

4.1.1. Polygons

A first example of figures built with P systems are regular polygons. In such polygons the length of the side is constant and the angle of deviation from the previous side is also constant. A simple calculus shows us that a deviation of $\delta = 360/n$ degrees allows us to built a regular polygon of n sides.

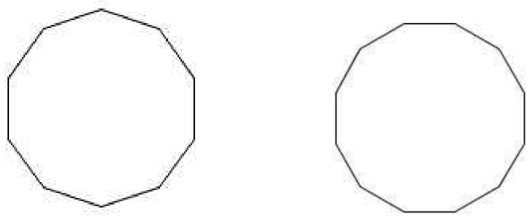


Figure 4: 10-polygon and 12-polygon

Figure 4 shows regular polygons of $n = 10$ and $n = 12$ sides obtained with $\delta = 36$ and $\delta = 30$ degrees. Obviously the number of steps are 10 and 12 respectively. The P system is the following

`Initial configuration:`	$[F\,W\,H]$
`Rule:`	$H \rightarrow [-F\,W\,H]$

4.1.2. Spirals

In mathematics, a spiral is a curve which emanates from a central point, getting progressively farther away as it revolves around the point. The concise mathematical definition is *the locus of a point moving at constant speed whose distance from a fixed point increases at a specific rate.*

An Archimedean spiral (a spiral named after the 3rd-century-BC Greek mathematician Archimedes) is the locus of points corresponding to the locations over time of a point moving away from a fixed point with a constant speed along a line which rotates with constant angular velocity. Equivalently, in polar coordinates (ρ, ω) it can be described by the equation $\rho = a + b\omega$ with real numbers a and b. Archimedes described such a spiral in his book *On Spirals*. It can be represented with the following P system:

`Initial configuration:`	$[F^n W H L]$
`Rules:`	$H \rightarrow [-F^n W L H]$
	$L \rightarrow LF$

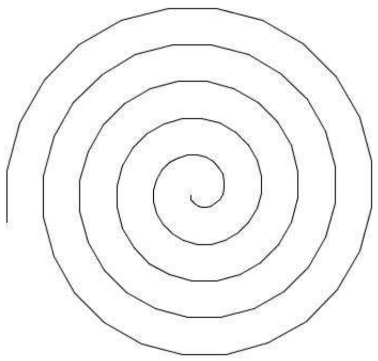


Figure 5: Archimedes' spiral

Figure 5 shows the representation of such Archimedes spiral for $n = 5$, length of $F = 0.01$, width $W = 1.0$, angle $\delta = 15$ and step 120.

The logarithmic spiral is a special kind of spiral curve which often appears in nature. It was first described by Descartes and extensively investigated by Jakob Bernoulli, who called it *Spira mirabilis*, "the marvelous spiral". Its equation in polar coordinates is $\rho = c^{\omega}$. It can be approximated by the P system

```
Initial configuration:  $[F^n W H L]$

              Rules:  $H \rightarrow [-F^n W L H]$
                      $L \rightarrow L M_1 F$
                      $M_1 \rightarrow M_2$
                      $\ldots$
                      $M_{i-1} \rightarrow M_i$
                      $M_i \rightarrow L$
```

Figure 6 shows the representation of such logarithmic spiral for $n = 10$, $i = 7$, length of $F = 0.001$, width $W = 1.0$ angle $\delta = 30$ and step 40.

4.2. Friezes

Another application of JPLANT for the graphical representation of restricted P systems with membrane creation is the design of friezes.

With the appropriate interpretation of the symbols, the following P system can be represented as a frieze based on right angles which has a flavor of Greek friezes. It can be extended horizontally in an unbounded manner.

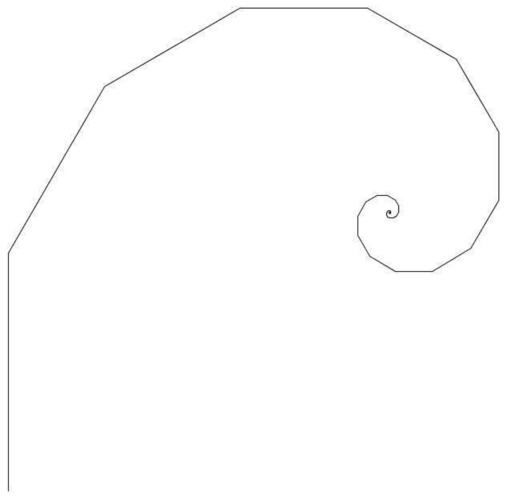


Figure 6: Logarithmic spiral

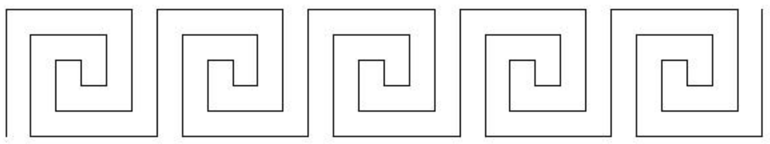


Figure 7: The first frieze

`Initial configuration:` $[F^5\, W\, H_1]$

`Rules:`	$H_1 \to [-\, F^5\, W\, H_2]$	$H_7 \to [+\, F\, W\, H_8]$
	$H_2 \to [-\, F^4\, W\, H_3]$	$H_8 \to [+\, F^2\, W\, H_9]$
	$H_3 \to [-\, F^3\, W\, H_4]$	$H_9 \to [+\, F^3\, W\, H_{10}]$
	$H_4 \to [-\, F^2\, W\, H_5]$	$H_{10} \to [+\, F^4\, W\, H_{11}]$
	$H_5 \to [-\, F\, W\, H_6]$	$H_{11} \to [+\, F^5\, W\, H_{12}]$
	$H_6 \to [-\, F\, W\, H_7]$	$H_{12} \to [+\, F^5\, W\, H_1]$

Figure 7 shows the representation of such frieze for length of $F = 0.5$, width $W = 1$ angle $\delta = 90$ and step 60.

Figure 8 shows a horizontally bounded frieze based on the Archimedes spiral.

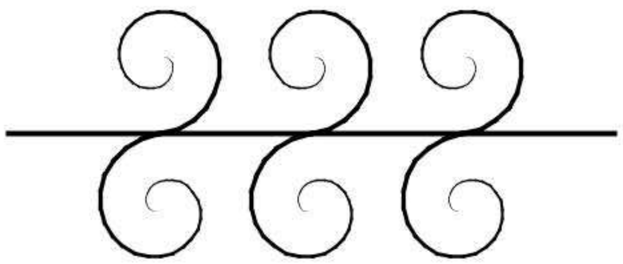


Figure 8: The second frieze

`Initial configuration:`	$[F^{300}\, W^{40}\, H_1\, I_1\, D_1]$	
`Rules:`	$H_1 \to [F^{300}\, W^{40}\, H_2\, I_2\, D_2]$	$I_1 \to I_2$
	$H_2 \to [F^{300}\, W^{40}\, H_3\, I_3\, D_3]$	$D_1 \to D_2$
	$H_3 \to [F^{300}\, W^{40}]$	$I_2 \to I_3$
	$L \to L\, F$	$D_2 \to D_3$
	$K \to K\, W$	$I_3 \to I_4$
	$I_4 \to [-^{11}\, F\, W\, L\, K\, D_4]$	$D_3 \to D_4$
	$D_4 \to [+\, F\, W\, L\, K\, D_4]$	

Figure 8 shows the representation of such frieze for length of $F = 0.01$, width $W = 0.1$ angle $\delta = 15$ and step 40.

4.3. Plants

Figure 9 shows the corresponding graphical representation of the ninth configuration of the P system presented in Section 2.1, where we fix a bottom-up orientation with a length $F = 1$, width $W = 2$ and an angle δ of 15 degrees.

Figure 10 represents four different trees obtained with JPLANT from the P system in Section 2.2.

5. Conclusions and Future Work

In this paper we have shown the suitability of P systems for modeling the growth of branching structures. It is our opinion that using membrane computing for this task could be an alternative to L-systems, the model most widely studied nowadays, for several reasons: the process of growing is closer to reality, since for example a plant does not grow by "rewriting" its branches, but by lengthening, widening and ramifying them; the membrane structure of P systems supports better and clearer the differentiation of the system into small units, easier to understand and possibly with different behaviors; the computational power of membrane systems can provide tools to easily simulate more complex models of growing, for example taking into account the flow of nutrients or hormones.

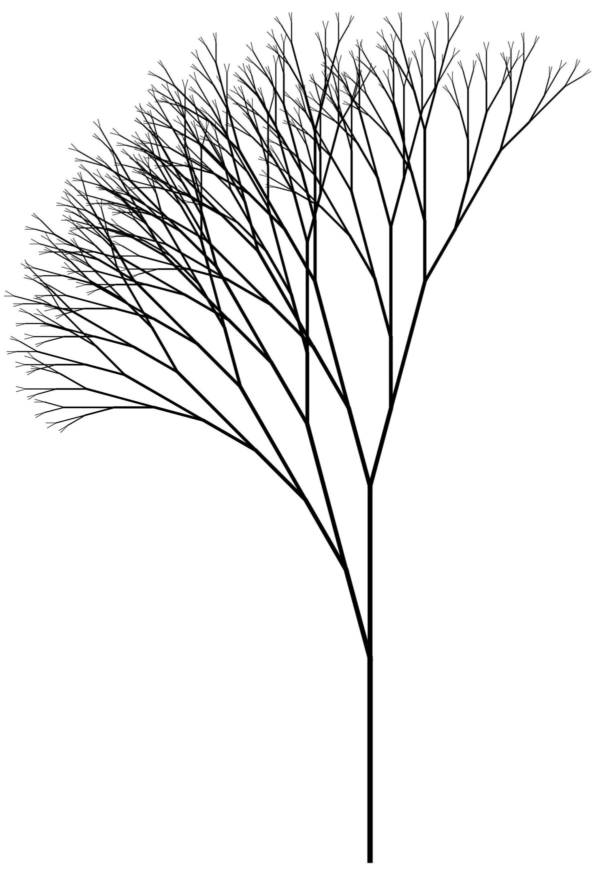


Figure 9: Tree

Nevertheless, it is still necessary a deeper study of several features of our proposed framework as compared with that of Lindenmayer systems. Two aspects that have to be investigated are the complexity of the models that can be constructed, and the computational efficiency in order to generate their graphical representation. On one hand, the use of the ingredients of membrane computing can lead to more intuitive models; on the other hand, we lose the linear sequence of graphical commands that characterize the parsing algorithm of L-systems.

From a theoretical point of view, one of the main drawbacks of the model is that it is extremely simple. Although the orientation of the paper belongs to the framework of membrane computing, the exclusive use of rules of type $a \to v$ and $a \to [v]$ miss the potential richness of expressiveness and computation of P systems. The following steps on this line should be devoted to the study of the graphical possibilities of P systems with more features, such as labels for the membranes (they can help to distinguish between different parts of a plant), the use of communication rules, allowing objects to cross the membranes of the system, division and/or dissolution rules, rules with cooperation, etc.

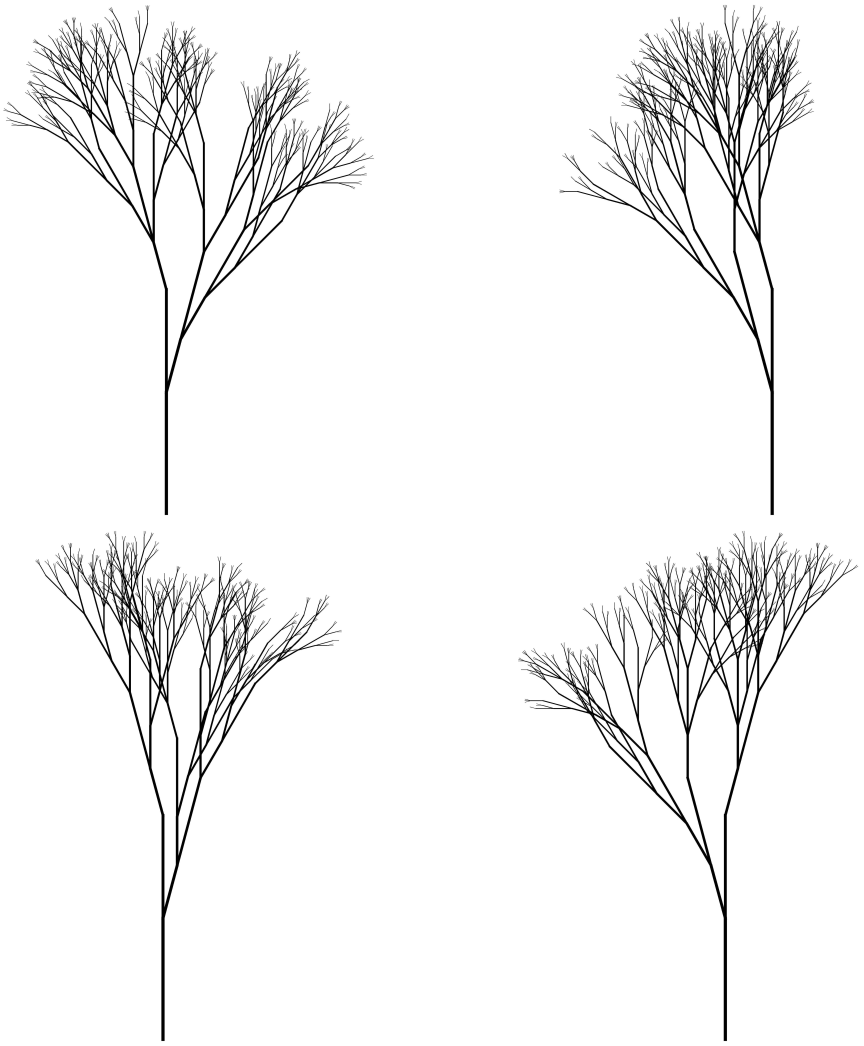


Figure 10: Four configurations

Acknowledgements

The authors acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the project of excellence TIC-581 of the Junta de Andalucía.

References

[1] ARDELEAN, I., CAVALIERE, M., Modelling Biological Processes by Using a Probabilistic P System Software, Natural Computing 2 (2) (2003), 173–197.

[2] GEORGIOU, A., GHEORGHE, M., Generative Devices Used in Graphics, in: A. Alhazov, C. Martín–Vide, Gh. Păun, (Eds.), Preproceedings of the Workshop on Membrane Computing, Tarragona, Spain, 2003, 266–272.

[3] GEORGIOU, A., GHEORGHE, M., BERNARDINI, F., Membrane-Based Devices Used in Computer Graphics, in: G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez (Eds.), Applications of Membrane Computing, Springer-Verlag, Berlin, Heidelberg, New York, 2006, 253–282.

[4] ITO, M., MARTÍN-VIDE, C., PĂUN, Gh., A characterization of Parikh sets of ET0L languages in terms of P systems, in: M. Ito, Gh. Păun, S. Yu (Eds.), Words, Semigroups, and Transducers, World Scientific, 2001, 239–254.

[5] LINDENMAYER, A., Mathematical models for cellular interaction in development, Parts I and II, Journal of Theoretical Biology 18 (1968), 280–315.

[6] MADHU, M., KRITHIVASAN, K., P Systems with Membrane Creation: Universality and Efficiency, in: M. Margenstern, Y. Rogozhin (Eds.), Proceedings of the Third International Conference on Universal Machines and Computations, Lecture Notes in Computer Science 2055 (2001), 276–287.

[7] OBTUŁOWICZ, A., PĂUN, Gh., (In search of) Probabilistic P systems, Biosystems 70 (2) (2003), 107–121.

[8] PESCINI, D., BESOZZI, D., MAURI, G., ZANDRON, C., Dynamical probabilistic P systems, International Journal of Foundations of Computer Science 17 (1) (2006), 183–204.

[9] PĂUN, Gh., Membrane Computing. An Introduction, Springer-Verlag, Berlin 2002.

[10] ROMERO-JIMÉNEZ, A., GUTIÉRREZ-NARANJO, M.A., PÉREZ-JIMÉNEZ, M.J., The growth of branching structures with P systems, in: C. Graciani-Díaz, Gh. Păun, A. Romero–Jiménez, F. Sancho-Caparrini (Eds.), Proceedings of the Fourth Brainstorming Week on Membrane Computing, Vol. II, Sevilla, Spain, 2006, 253–265.

[11] ROMERO-JIMÉNEZ, A., GUTIÉRREZ-NARANJO, M. A., PÉREZ-JIMÉNEZ, M. J., Graphical modelling of higher plants using P systems, in: H. J. Hoogeboom, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), Membrane Computing, Seventh International Workshop, WMC 2006, Lecture Notes in Computer Science 4361, 2006, 496–506.

[12] SMITH, A.R., Plants, Fractals and formal languages, Computer Graphics 18 (3) (1984), 1–10.

[13] The P Systems Web Page `http://ppage.psystems.eu/`

Author Index

Aman, Bogdan, 17

Bandyopadhyay, Anirban, 9

Ceterchi, Rodica, 29
Ciobanu, Gabriel, 17

Franco, Giuditta, 11
Freund, Rudolf, 43

Gutiérrez-Naranjo, Miguel A., 87

Krassovitskiy, Alexander, 53
Krishna, Shankara Narayanan, 15

Neary, Turlough, 65

Pérez-Jiménez, Mario, 29

Ramlan, Effirul I., 75
Riscos-Núñez, Agustín, 87
Rivero-Gil, Elena, 87
Rogozhin, Yurii, 53
Romero-Jiménez, Álvaro, 87

Tomescu, Alexandru Ioan, 29

Verlan, Sergey, 43, 53

Zauner, Klaus-Peter, 75