Hendrik Jan Hoogeboom
Gheorghe Păun
Grzegorz Rozenberg
Arto Salomaa (Eds.)

LNCS 4361

Membrane Computing

7th International Workshop, WMC 2006
Leiden, The Netherlands, July 2006
Revised, Selected, and Invited Papers

Lecture Notes in Computer Science 4361

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Hendrik Jan Hoogeboom Gheorghe Păun
Grzegorz Rozenberg Arto Salomaa (Eds.)

Membrane Computing

7th International Workshop, WMC 2006
Leiden, The Netherlands, July 17-21, 2006
Revised, Selected, and Invited Papers

Springer

Volume Editors

Hendrik Jan Hoogeboom
Leiden Center of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
E-mail: hoogeboom@liacs.nl

Gheorghe Păun
Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania, and
Research Group on Natural Computing
Department of Computer Science and AI
Seville University, 41012 Seville, Spain
E-mail: george.paun@imar.ro, gpaun@us.es

Grzegorz Rozenberg
Leiden Center of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
E-mail: rozenberg@liacs.nl

Arto Salomaa
Turku Centre for Computer Science (TUCS)
Leminkäisenkatu 14, 20520 Turku, Finland
E-mail: asalomaa@cs.utu.fi

Library of Congress Control Number: 2006939014

CR Subject Classification (1998): F.1, F.4, I.6, J.3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-540-69088-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-69088-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11963516 06/3142 5 4 3 2 1 0

Preface

The present volume contains a selection of papers presented at the Seventh Workshop on Membrane Computing, WMC7, which took place in Leiden, The Netherlands, during July 17–21, 2006. The first three workshops on membrane computing were organized in Curtea de Argeş, Romania – they took place in August 2000 (with the proceedings published in *Lecture Notes in Computer Science*, volume 2235), in August 2001 (with a selection of papers published as a special issue of *Fundamenta Informaticae*, volume 49, numbers 1–3, 2002), and in August 2002 (with the proceedings published in *Lecture Notes in Computer Science*, volume 2597). The next three workshops were organized in Tarragona, Spain, in July 2003, in Milan, Italy, in June 2004, and in Vienna, Austria, in July 2005, with the proceedings published as volumes 2933, 3365, and 3850, respectively, of *Lecture Notes in Computer Science*.

The 2006 edition of WMC was organized (and supported) by Lorentz Center, Leiden, under the auspices of the European Molecular Computing Consortium (EMCC). Special attention was paid to the interaction of membrane computing with biology, focusing both on the biological roots of membrane computing and on applications of membrane computing in biology and medicine. Furthermore, the meeting was planned also as an event promoting the interaction and cooperation between the participants (e.g., the workshop was one day longer than usually, with afternoons devoted mainly to joint work).

The pre-proceedings of WMC7 were published by the Institute of Advanced Computer Science (LIACS) of Leiden University, and they were available during the workshop. Each paper was refereed by two members of the Program Committee. As an indication of the healthy state of this research area, it is worth noting that both the number of submitted papers and the total number of contributing authors were bigger this year than last year, while the number, the variety, and the intricacy of applications (mainly in biology and medicine) also increased substantially. These observations are confirmed by the present volume. Most of the included papers were significantly modified according to the discussions that took place during the workshop.

The volume includes all the invited talks (seven this time) – more than for any proceedings of previous editions of WMC; moreover, this time the invited talks were chosen in such a way as to reflect the relationships of membrane computing to biology and medicine. Consequently, this volume is a faithful illustration of the current state of research in membrane computing (a comprehensive source of information about this fast-emerging area of natural computing is the Web page `http://psystems.disco.unimib.it`).

The Program Committee consisted of Matteo Cavaliere (Trento, Italy), Erzsébeth Csuhaj-Varjú (Budapest, Hungary), Marian Gheorghe (Sheffield, UK), Hendrik Jan Hoogeboom (Leiden, Netherlands) – Co-chair, Oscar H. Ibarra

(Santa Barbara, USA), Natasha Jonoska (Tampa, Florida), Shanhara Narayanan Krishna (Bombay, India), Gheorghe Păun (Bucharest, Romania, and Seville, Spain) – Co-chair, Mario J. Pérez-Jiménez (Seville, Spain), and Claudio Zandron (Milan, Italy).

The editors are indebted to the members of the Program Committee, to all participants in WMC7, and in particular to the contributors to this volume. Special thanks go to Lorentz Center, Leiden, for the perfect organization of the workshop, and to Springer for the efficient cooperation in the timely production of the present volume.

November 2006

Hendrik Jan Hoogeboom
Gheorghe Păun
Grzegorz Rozenberg
Arto Salomaa

Table of Contents

Invited Lectures

Regular Papers

Biological Roots and Applications of P Systems: Further Suggestions

Ioan I. Ardelean

Institute of Biology, Romanian Academy
Centre of Microbiology, Splaiul Independentei 296
P.O. Box 56-53, Bucharest 060031, Romania
ioan.ardelean@ibiol.ro

Abstract. P systems offer the possibility to appropriately describe discrete processes performed by: i) single objects: catalytic molecules (= enzymes), supramolecular structures (MscL, porins, ionic channels etc.), single cells, and ii) a small number of objects occurring in the sample, e.g., several mechanosensitive channels occurring within a membrane patch. Thus P systems could offer the possibility to capture and model the plethora of experimental data obtained in the emerging and rapidly growing field of single cell or single molecule or atom studies.

Furthermore, it is suggested that *in vitro* implementation of P systems could be done by the use of artificial membranes, a step forward computations with artificial membranes.

1 Introduction

In the framework of the dialog between P systems and (Micro)Biology the aim of this paper is to add further arguments that i) biological roots of P systems are the computations performed by living cells when they carry out chemical reactions and physical processes, and ii) that discrete mathematics is very appropriate to describe discontinuous biological processes.

Furthermore, I claim that P systems are very appropriate to model the function of single objects (cell, channel, enzyme, etc.) and suggest that *in vitro* implementation of P systems could be done by the use of artificial membranes, and this can be a step forward computations with artificial membranes.

2 Biological Roots

At the beginning of experimental research in biology the dominant opinion was that inside living cells there is no chemical reaction or physical processes (Mayer, 1998). When the experimental biology significantly developed (in the first part of XIXth century), it became evident for all scientists that chemistry and physics can be used to study living organisms because: i) cells are composed of (different) chemicals, and ii) within the cells the chemicals follow chemical reactions and physical transformations which should obey the laws of chemistry and physics.

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 1–17, 2006.

The use of Chemistry and Physics to study Biology was not an easy and smooth approach; for example, a historical debate concerned the usefulness of thermodynamics in the study of biology. Thermodynamics introduced, in the first half of XIXth century, the key concept of energy, as the quantitative expression of the capacity of any system for doing work and overcome resistance (for more details see Ciures and Mărgineanu, 1970, Morowitz, 1972; Mărgineanu, 2001).

There were authoritative voices claiming that living matter does not obey the second law of thermodynamics, whose usefulness for biology was under question until the emergence of so called non-equilibrium thermodynamics, or thermodynamics of irreversible processes (Glandsdorff and I. Prigogine, 1971).

Non-equilibrium thermodynamics can scientifically explain why it is physically possible the occurrence of irreversible processes such as those within living organisms which consume chemical energy in the form of nutrients, perform work, and excrete waste as well as give off heat to the surroundings (for more discussions, see Glandsdorff and Prigogine, 1971, Morowitz, 1972; Mărgineanu, 2001).

In the framework of P systems it seems appropriately to recall that Prigogine who won the Nobel Prize in Chemistry in 1977 for his contributions to non-equilibrium thermodynamics, stressed on the important contribution of Alan M. Turing to the background of non-equilibrium thermodynamics. Working on a new mathematical theory of morphogenesis based on showing the consequences of non-linear equations for chemical reactions and diffusions, Turing (1952) was the first to notice the possibility of bifurcations in chemical reactions, ("Turing bifurcation"), which are essential for non-equilibrium thermodynamics (Glandsdorff and Prigogine, 1971).

So far, mathematical tools used in Biology via Physics and Chemistry are differential equations. However the emergence of P systems in 1998-2000 (Păun, 2000; 2001; 2002) opened a new era in the use of Mathematics in Biology because: a) It was for the first time the biological realities (occurring mainly across biological membrane), were used as raw material to abstract a calculus; b) it was a further incentive for using the discrete Mathematics for the description/modelling of discontinuous processes occurring within living cells.

The ontological fundament of P systems (apart from the inspiration of its initiator, the chance etc.) seems to me to be the fact that the processes occurring across biological membranes are genuine natural calculi performed by living cells.

Thus the claim is that chemical reactions and physical processes (e.g., transport) within a cell are computing processes.

The emergence and flourishing of cell biology with special emphasis on membrane biophysics, biochemistry, and molecular biology, generated knowledge concerning discontinuous processes occurring across, within, and at biomembranes (Alberts et al., 1994), knowledge that sustained the emergence of membrane computing (P systems).

The introduction of chemistry and physics into biology mainly in the XXth century produced a wealth of knowledge concerning the fluxes of mater and energy within a living cell. Thus it becomes evident that any biological system,

for instance, a living bacterium, is an open system which exchanges mater, energy, and information with its surroundings medium. These fluxes are possible because some *functional* proteins are involved not only in processing matter and energy but also in the flow of information within a cell (Bray, 1995). The flux of information is illustrated, rather metaphorically than mathematically, by the flux of genetic information from DNA to polypeptide during protein biosynthesis, the function of axons and signal transduction, including chemotaxis (Mayer 1998; Bray 2002; Szurmant and Ordal, 2004; Martinez-Antonio et al., 2006).

Interesting for our topic, the biological community accepted that the function of a neuron, including the generation and propagation of action potentials (AP), is a computation performed by the cell (Segev and Schneidman, 1999; Alle and Geiger, 2006).

The basic equations (see bellow) and quantitative description of the generation and propagation of action potentials (spike) is due to Hodgkin and Huxley who won Nobel Prize in 1963. The authors supposed that in the nerve cell membrane there are distinct ionic channels which allow the selective passage of either sodium or potassium ions (see bellow). The experimental proof of the occurrence of these channels came later on. Beyond its interest *per se*, this first quantitative description of a major biological event, with a consistency which previously seemed reserved to physics, has the merit of showing how a proper metaphor can evolve into a truly mechanistic description (Mărgineanu, 2001).

This opinion is important for our topic, because:

- the so far work in P systems pushes the metaphor (cell computer, cell computing, membrane computing) toward a mechanistic description;
- in the framework of P systems ions movement across biological membrane (either via ionic channels or symport/antiport) is largely studied, while the biochemical background of neuron function is the fact that axons typically possess a high density of voltage-dependent, fast activated Na^+ channels, as well as more slowly activated K^+ channels;
- moreover, recently neuron function started to be copiously studied by the specific tools and concepts of P systems (Ibarra et al., 2006; Ionescu et al., 2006; Chen et al., 2006a, b, c, d).

The computation performed by the neuron concerns (Alle and Geiger, 2006): i) the ability to control the number of APs, ii) the ability of at least some neurons to control the timing of the spikes (Carr and Konishi, 1988), iii) subthreshold membrane potentials (those potentials which per se do not generate an AP), which have a role in the regulation of synaptic transmission (Alle and Geiger, 2006).

The occurrence of this analogue coding and AP coding (number and timing of AP) is likely to enhance information capacity of synapses and may increase the computational power of a network of neurons (Alle and Geiger, 2006). These three ways to increase the computational power of (a network of) neurons could receive more attention from P systems, for the benefits of both parts, I believe.

Another important aspect for P systems is the type of mathematics used so far to model AP. As reviewed by Segev and Schneidman, (1999), following

the theoretical study of Hodgkin and Huxley most of the models of axon have treated the generation and propagation of the AP using deterministic partial differential equations although it is known that the underlying mechanism for AP generation is the opening and closing of thousands of individual ion channels, each of which is stochastic rather than deterministic (Segev and Schneidman, 1999). So far, based on partial differential equations of Hodgkin and Huxley, the scientists developed a deterministic H&H model and a stochastic H&H model (for more details see Segev and Schneidman, 1999) for spike generation.

For a very large number of channels (12,000 and 3,600 for sodium and potassium, respectively) both deterministic and stochastic models give similar estimations, whereas for, e.g., 12,000 and 3,600 for sodium and potassium channels per patch, respectively, some differences started to appear. With the interest of P systems to develop a hybrid metabolic algorithm capable of mixing the deterministic and stochastic paradigms together (Bianco and Fontana, 2006) it seems rational to expect contribution from P systems to also model the AP generation.

Important for P systems is the question whether or not the discrete formalism would describe more accurately the process of spike generation, by one neuron or by a limited number of neurons. Can the metaphoric membrane computing evolve into a scientific truth, towards a mechanistic description? Based on the work in the field of P systems, it seems appropriate to conclude that this metaphor is actively evolving, as it seems natural to assume that discontinuous processes can be appropriately described/modelled by discrete mathematics.

In the following I will briefly present two membrane-based biological processes which I believe could be described in the (near) future as membrane computing processes: transport of ions and molecules across biological membranes and chemotaxis (Ardelean, 2002).

The transport of ions and molecules across biological membranes is fundamental for cell structure and function. There are several ways to carry out this transport (Saier, 1999). Here we will briefly focus on symport/antiport, because 1) this type of transport received significant attention in P systems (Păun, 2002), symport and antiport being nice examples of how bacterial cells manage the developmental rules; 2) the P system tools used to describe symport and antiport are appropriate to describe the function of, e.g., sodium and potassium channels which sustain the function of neuron, the computing device recognized by biologists; 3) the proteins involved in symport and antiport (as well as other proteins – see below) after extraction from natural membranes can be incorporated in artificial lipid membranes, while retaining the activity; this is a way for a possible *in vitro* implementation of P systems (see below).

Symport is characterized by the fact that both ions are transported in the same direction, whereas by antiport one ion is transported inside the cell while the other ion is simultaneously transported in the opposite direction, outside the cell. Both systems of transport, symport and antiport, are used by bacteria; the symport of protons with different substances needed for bacterial growth are very well documented (Jung, 2001). For example, *Escherichia coli* uses the symport of protons with lactose, arabinose or galactose.

However, the proton is not the only type of ions used with antiport systems; sodium ions are also used for the symport of substances such as melobiose and proline. When it comes to antiport, one classical example is the proton/sodium antiporters found in many bacteria, their major function being in maintaining a rather constant concentration of both protons and sodium ions inside the cell (Padan et al., 2001).

With the development of the mutual interplay between P systems and biology, probably, the future is not so far when the image of a single (bacterial) cell counting/computing the input and the output of ions and molecules via symport/antiport proteins will be no more a metaphor but a truly mechanistic description.

Chemotaxis, the movement of a bacterium (or other types of cells) toward a needed chemical factor/item (called attractant) or away from a dangerous chemical (called repellent) is one example of cell behavior involving fluxes of matter, energy, and information. This behavior is essential for bacterial survival.

Chemotaxis allows the bacterium to actively move towards the needed substance (or more precisely, towards the increase in the gradient concentration of that substance); for example, oxygen respiring bacteria move towards (optimal) oxygen concentrations (= aero taxis).

Chemotaxis also allows the bacterium to try to escape from a substance that is toxic for it (a poison, for example). In a living bacterium (cell, in general) the movement towards a factor (e.g., molecular oxygen) is related to the orientation toward that factor. For example, magnetotactic bacteria contain inside the cell specific particles/structures called magnetosome which consist of magnetic iron mineral particles enclosed within a membrane. The specific functional characteristic of magnetotactic bacteria is magnetotaxis, the orientation along the Earth geomagnetic field lines (Blakemore, 1975). Magnetotaxis is determined by the presence of magnetosomes; dead cells containing magnetosomes also align along the geomagnetic field lines whereas living MTB with no magnetosomes, do not align.

Chemotaxis is important for P systems (and natural computing in general) because the coordinated movement and binding of molecules within and chemical reactions across the cell membrane sustain the emergence of macroscopic (at the level of several centimeters) information-oriented behavior. The claim is that a taxi is a membrane-computing process whose study would be significantly improved by adding P systems to the plethora of methods and concepts which are contributing to the blossom of this topic.

The overall process of chemotaxis involves the following four integrated stages, very shortly presented here (for more biochemical essential details, see Armitage, Backer et al., 2006; Szurmant and Ordal, 2004; Ardelean and Besozzi, 2006).

1. Signal recognition and transduction is performed by receptor proteins. Receptor proteins are usually transmembrane, multidomain proteins, which contain a sensing domain that interact with the environmental signal (oxygen concentration, for example) and, through series of chemo-physical changes induce further changes in the signalling domain, changes which further enter

the second stage, excitation. Interestingly, in *E. coli* as in other cells where the receptors have an extra-membrane domain, the localization of the receptors is around the poles of the cells, and not uniformly over the whole cells surface (see below)
2. Excitation. The main protein of this stage is an intracellular sensor kinase called CheA kinase (CheA), which autophosphorylates, and the phosphoryl group is subsequently transferred from CheA to CheY. In the phosphorylated form, as CheY-P, this protein binds to an assembly of proteins called the "switch", at the base of the flagellar motor, thus controlling the direction of the flagellar rotation.
3. Adaptation. This is very important for bacterial taxis because requires the ability to recognize when the bacterium is moving in the wrong direction, i.e., away from the higher attractant concentration (Szurmant and Ordal, 2004). To do that, a "memory" is required that is able to indicate whether higher or lower concentration are being reached.
4. Signal removal means the biochemical removing of phosphate from CheY-P; the resulting dephosphorylated form (CheY) binds no more on the flagellum.

In my opinion, adaptation and signal removal could offer fruitful space for cross talk between biology and P systems.

3 Applications of P Systems

So far, the main applications of P systems are in biology, computer sciences, linguistics, and membrane software (Ciobanu et al., 2006). Here, I will shortly present those application in biology to which I had the chance to work on together with P system scientists and other computer scientists.

3.1 Membrane Proteins: Terminal Respiratory Enzymes and Photosystem II

We tried to compare the probabilistic mathematical model with the biological reality, indicating how one can use the P systems framework to simulate the process of respiration in *Escherichia coli* and *Synechocystis* PCC 6803, the corresponding proton pumping by cytochrome *c* oxydase in *Anacystis nidulans*, the interplay between oxygen consumption and oxygen production by photosystem II (PSII) in *Synechocystis* PCC 6803 even in the presence of a specific synthetic inhibitor of PSII. We also showed how to interpret the obtained results in a way to infer useful results for biologists (Ardelean and Cavaliere, 2006; Cavaliere and Ardelean, 2006).

This work has been done in the framework of P systems because we believe that the emergence of P systems together with its cross talk with biologists could be for biology as important and fruitful as it was, and still it is, the introduction of physics and chemistry in biology, almost two centuries ago.

We have presented a comparison between the mathematical model (and the software realized) and the *real world*, trying to establish a link between the mathematical framework, the simulator realized (Cavaliere, 2003), and the biological

reality. We introduced new concepts in the P system area such as the *availability* of a chemical reaction, the *activity rate* of a catalyst, and the possibility for a catalyst to be at same time *active* or *not active*.

I will briefly recall some more important achievements concerning respiration (for more details and modeling of proton pumping and the interplay between oxygen consumption and oxygen production by PSII in cyanobacteria, please see Cavaliere and Ardelean, 2006).

Respiration is the biological process that allows the cells (from bacteria to humans) to obtain energy. In short, respiration promotes a flux of electrons from electron donors to a final electron acceptor, which in most cases is the molecular oxygen.

In *Escherichia coli*, as well as in other bacteria, the cell ability to consume molecular oxygen during the respiration is determined by the presence of two different enzymes that catalyze the final step of respiration: the reduction of molecular oxygen with protons and electrons. In *Escherichia coli*, these two terminal oxydases (enzymes) – called *terminal* because they are the last components of the respiratory electron transport pathways – are *cytochrome bd* and *cytochrome bo*.

For example, it is know that at low oxygen concentration in the growing medium (lower than about 40% of oxygen saturation) the cytochrome *bd* oxydase is responsible for the entire respiratory activity of the cells; in other words, the flux of electrons to molecular oxygen proceeds 100% through the cytochrome *bd* oxydase. At high oxygen concentration in the growing medium (this means in between 90% and 100% of oxygen saturation), the cytochrome *bo* oxydase is responsible for almost the entire respiratory activity of the cells. Furthermore, in between 40% and 90%, the two types of terminal oxidases contribute together to the respiration of the cell.

We used the probabilistic P systems software to model the respiratory oxygen consumption when only cytochrome *bd* oxydase or only cytochrome *bo* oxydase or both terminal oxydases are active in intact cells. These simulations are in agreement with experimental results sustaining our claim that the software could be used to perform *in silico* experiments in order to estimate, based on new experimental results on *E. coli*, the contribution of each terminal oxidase to the overall respiratory oxygen consumption in given environmental conditions. We also believe that the software can be used to model the respiration in other bacteria having two terminal oxiadses, even if they have different affinities for oxygen than the terminal oxidases in *E. coli*.

In the near future, we plan to model other biological processes with more biological details considering the concept of *affinity* introduced in our simulator by the so-called probability to win and the concept of availability of a rule, modeled in the software by the probability of a rule to be available; in particular it would be very useful to add to the simulator the ability to change, in run-time, the values of some of the biological parameters considered (like the affinity of an enzyme, that, for example in *Escherichia coli*, changes according to the concentration of molecular oxygen in the substrate); in this way, we can

improve at the same time the simulator and the mathematical model presented, as well as making possible new applications in microbiology. In this respect, the requirement/ability of the P system software to pay attention to every single occurrence of each molecule of either cytochrome *bd* oxydase and cytochrome *bo* oxydase, opens the question concerning the usefulness of this software for modeling the activity of a single molecule, a single cell, etc., which is a major trend in nowadays science (see further prospects below).

3.2 Membrane Proteins: Mechanosensitive Channels

The activity of mechanosensitive channels of large conductance (MscL, in short) in cellular membranes was modeled within the framework of P systems (Ardelean et al, 2006), based on the opinion (Ardelean, 2003) that discrete mathematics could be more appropriate than continuous mathematics to describe non-continuous molecular events (such as channels opening and closure).

Mechanosensitive channels are protein-based channels gated by mechanical forces. In Gram-positive and Gram-negative bacteria, MscL is located in the cell membrane. This location in bacteria can be correlated to its physiological function, the protection against severe osmotic downshifts. The major role of Msc under osmotic downshift is to allow the rapid exit of different chemicals (ions and rather small molecules), and hence the sudden decrease of the osmotic pressure inside the cell. Thus, by the opening of MscL the osmotic pressure inside the cell approaches the osmotic values of the extracellular medium. This event is fundamental for bacterial cell because, when the difference between osmotic pressure inside the cell and osmotic pressure outside the cell is too large, the integrity of the cell can be damaged by disruption of cell wall and plasma membrane, followed by cell death.

In the paper (Ardelean et al., 2006) we defined *in vitro* and *in vivo* distinct models that consist of some basic components: an environment, a region, and a membrane tension, which naturally correspond to essential aspects of MscL activity. We also introduced probabilities associated to evolution rules in order to achieve a closer resemblance to biological reality. Moreover, we defined evolution rules according to *in vitro* and *in vivo* different environmental events. With *in vitro* model we focused on a single mechanosensitive channel (MscL) thus arguing that P systems are appropriate to describe single items, single events. The study of a single molecule, cell, etc. is a major trend in nowadays science, including microbiology and P systems could be the mathematical formalism of this growing trend (see further prospects below). We claimed that *in vitro* model can be easily extended to describe and simultaneously analyze multiple occurrences of MscL. We gave some notes about *in silico* simulations of the *in vitro* model, using *EdnaCo*, a complex systems simulator that can be used as a distributed discrete-event simulation environment (Garzon et al., 2000), and showed some results, such as the emergent behavior over time of membrane tension, conductance, and current of channels.

In vitro and *in vivo* P models might propose a platform for the integration of the data obtained on MscL in prokaryotes. Thus, we are actively expecting that

the further refinement of our models (also by means of the software environment used to run simulations), would accelerate the integration between *in vitro* and *in vivo* results. Moreover, with the explosion of molecular biology and the increase in the quantity and quality of data obtained by high throughput technologies, there is a trend in nowadays biology to pass from a reductionistic approach to an integrative approach (Palsson, 2000), either at supermolecular level (Hartwell et al., 1999) or at systemic level (Palsson, 2000; Kitano, 2002). Indeed, the reductionistic approach and the systemic, integrative approach, are today on the same side of the barricade, a totally different position than four-five decades ago when both approaches started to flourish in biology (Mayer, 1998). In this perspective, the correlation between *in vitro* and *in vivo* results represents one of the most important trends in the biological research, and the models mentioned in this chapter could give an important contribution to it.

Moreover, I put forward the claim that P systems could be the framework to develop the appropriate mathematical formalism for a cross talk between the reductionistic approach and the systemic, integrative approach (the further contribution from continuous mathematics is not excluded at all!).

4 Further Suggestions for P Systems

The basic suggestion concerns the need to continue the use of P systems to model discrete, discontinuous processes in the living world.

So far, the scientist used differential and integral calculus to describe continuous processes as well as discrete, discontinuous processes in biology (or in other sciences). A common strategy to figure out such equations consists in writing down equilibrium conditions for *infinitely small* physical units such as time units, dt, and spatial volume units, ds (Bianco et al., 2006); the scientist claim for the validity of this approach takes into account the assumption that the system is composed of a great number of undistinguishable particles (for more discussions see Mărgineanu, 2001).

In my opinion, the success of differential and integral calculus in modeling discrete processes (e.g., the kinetic of enzymatic reactions, ionic processes at the neuron plasma membrane, etc.) was based – apart from the above argument – on the fact that no discrete mathematical approach has been used to model biological processes. Fortunately, since the birth of P systems several interesting approaches have been already published on the use of discrete mathematics to model discrete biological processes.

Manca introduced the P metabolic algorithm (PMA) whose (main) principles are the followings (for more details, see Manca, 2006): i) rules compete for object populations; ii) objects are allocated to rules according to a mass partition principle; iii) partition factors are determined by reaction maps, and iv) a "metabolic rule" r consumes/produces integer multiples of a reaction unit u_r which generalizes the notion of molar unit. (Manca, 2006).

Metabolic P systems have several computational advantages with respect to differential models, but their most important aspect is their direct biological

meaning and their structure where the reaction level and the regulation level are clearly interconnected, but separated (Manca, 2006). PMA were successfully used in simulations concerning the evolution of several relevant biological processes such as Prey-predator Lotka-Voltera dynamics, leukocyte selective recruitment in immune response, protein kinase C activation, circadian rhythms, mitotic cycle, etc. (for more details, see Manca, 2006, and the references herein).

In turn, Pérez-Jiménez (2006) proposed a mesoscopic approach which is more tractable than the microscopic chemistry, but it provides a finer and better understanding than macroscopic chemistry modeled by ordinary differential equations. A deterministic waiting times algorithm has been introduced, based on the fact that *in vivo* chemical reactions take place in parallel in an asynchronous manner. The strategy has been illustrated with the simulation of two important biological phenomena: the epidermal growth factor receptor signalling cascade (Pérez-Jiménez and Romero- Campero, 2005) and FAS-induced apoptosis. The simulations performed by the authors show good correlations with both the data reported in the literature and simulations based on ordinary differential equations (more details, in Pérez-Jiménez, 2006).

All these results show that P systems can successfully compete with ordinary differential equations in producing good simulation of biological processes; and probably this is only the beginning.

However, even these approaches were done assuming a great number of (almost) undistinguishable particles/objects (e.g., enzymatic molecules, etc.), a ground where the competition with continuous mathematics appears to be hard.

In my opinion, P systems should “attack" (just to use o common concept in military art, or in the game of GO) on the territory where differential and integral calculus are not valid at all: the space of a single event, single occurrence of a catalyst (enzyme, mechanosensitive channel etc.).

The chance is that in nowadays science there is a strong trend toward the study of single events or objects (molecule, cell, etc.). To argue that, I will briefly focus on microbiology. Single cell microbiology (SCM) is a trend in microbiological sciences which allows the study of an individual cell from a population (Brehm-Stecher and Johnson, 2004). The development of SCM roots in the technical advances in other sciences (mainly physics and biochemistry) where special tools were developed. The advances in SCM are already copious and already a first authoritative review has been written on this topic (Brehm-Stecher and Johnson, 2004). It is beyond the scope of this contribution to present the state of the art of SCM; I will mention only a few achievements in SCM, which, in my opinion, are very appropriate to be further modelled by P systems.

a) SCM allows the observation of discontinuous and dynamical processes within living (bacterial, yeast, etc.) cells with high spatial and/or temporal resolution. An example is the specific distribution at the cell surface of some receptors (receptor clustering) involved in chemataxis. More precisely, it started to be shown that the protein acting as receptors in bacterial chemotaxis are not distributed uniformly at the surface of the *E. coli* cell, the main localization being at one pole of the cell (Bray, 2002) – it should be reminded

that *E. coli* is an almost cylindrical cell. This polarized distribution of protein receptors in space is an on growing topic in microbiology and it could be interesting for P systems at least as a new type of communication rules, in which the position, the density of the interacting "letters" (symbols) is essential for the rule to occur.

The interest and the power of P systems for the exploration of space is illustrated by the proposal of extended P systems able to perform a global description of membrane proteins populations, in order to take care of the synergic work of many membrane proteins and the related effect for cell's life (Besozzi and Rozenberg, 2006). These developments could be a further challenge for P systems to develop their applications in those domains where spatial distribution of objects is very important: collective sports (football etc.), military strategy, and games such as chess, GO, etc.

b) SCM allows the interplay between microscopic (in the sense of biology), mesoscopic, and macroscopic properties of microbial population, with emphasis on the cellular origins of mesoscopic and macroscopic properties. For example, coordinated movement of cells (e.g., traveling waves, whirls, and jets) within population of myxobacteria or *Bacillus subtilis* have been studied at cellular level integrated in the population level (more details, see Brehm-Stecher and Johnson, 2004). Similar movements have been described in magnetotactic bacteria whose study of movement of each individual cell still waits. So far, classical microbiology has traditionally been concerned with and focussed on the studies at the population level. Nowadays, with the emergence of SCM, microbiology faced the same problem: the connection between macroscopic (= population) level behavior with microscopic (= cellular, in biology) level. Long time ago, thermodynamics solved this problem by differential and integral calculus. The nowadays answer of microbiology can be based on the use of P system to describe discrete processes based either on single, small or huge (statistically relevant) numbers of cells. Furthermore, with the development of appropriate tools and techniques to study discrete realities such as a single individual microbial cell, there is the need for discrete models whose appropriate description seems naturally to be founded on discrete mathematics, P systems appearing as a very powerful candidate for this task.

c) The analytic progress of microbiology towards the study of single individual cells raised new theoretical problems. Because of the fact that the experiment and its control cannot be carried out on the same individual cell, it is impossible to be sure that the observation/measurements itself does not affect the cell. This statement is the biological equivalent of Heisenberg's "uncertainty principle" (for more details, see Brehm-Stecher and Johnson, 2004).

d) SCM has a large significance not only in basic science but also in applied science, mainly with respect to microbial heterogeneity related to antibiotic and biocide resistance, productivity and stability of industrial microbial based biotechnologies or the potential of pathogens to cause disease (see more details in Brehm-Stecher and Johnson, 2004, and references therein).

e) Single molecule kinetics, a growing topic in biochemistry (Shi et al., 2006) could be performed not only *in vitro* but also *in vivo*, in a single cell approach, and P systems could be more appropriate in modeling the single molecule kinetics than Michaelis-Menten differential equations, whose basic assumption that enzyme-substrate concentration are continuous variables seems not to be valid at the microscopic level.

Thus, it seems rational to conclude (and expect for the near future) that P systems, could offer the appropriate software for the description of discrete biological processes (such as those related to SCM), and that membrane computing will be used to model the experimental data obtained on single cells (molecules, etc.) not only in basic science (e.g., mechanosensitive channel behavior, enzymes, enzymatic cascade, etc.), but also in applied science such as the contribution of the sensibility of each individual cell from a population against a given substance, for the design of biocides.

In conclusion, I suggest that:

1. P systems are invaluable to describe discrete processes performed by: i) single objects: catalytic molecules (= enzymes); supramolecular structures (MscL, porins, ionic channels, etc.), single cells, and ii) a small number of objects occurring in the sample, e.g., several mechanosensitive channels occurring within a membrane patch. Furthermore, P systems could be valuable tools to describe discrete processes performed by a statistically significant number of undistinguishable particles.
2. The use of P system to write a software able to manage the wealth of information obtained from different types of arrays: DNA- arrays, protein-arrays, enzymes. One requirement with these arrays is the rapid and correct process of information produced by them. Again, a reading tip leaded/controlled by a discrete programme based on discrete mathematics would be more successful to monitor a discrete process (arrays function) than a classical programme.
3. The already started interplay between P systems and ordinary differential equations (Pérez-Jiménez, 2006; Manca, 2006) supports the proposal to develop a discrete formalism for discrete processes performed by single molecules/structures as shown for mechanosensitive channel in bacteria (Ardelean et al., 2006).
4. Towards an *in vitro* implementation of P systems. The progresses made in the last four decades in incorporating different biological molecules into artificial membranes(e.g., black lipid membrane – BLM) has lead to major progresses in understanding their *in vivo* function (Ottowa and Tien, 2002). In the next table there are presented some reconstituted systems within artificial membranes, with emphasis on original application for a possible *in vitro* implementation of P systems.

The incorporation of different active (mainly) protein molecules in artificial membranes opens the possibility to move objects across these membranes, and to perform a calculus. In Figure 1 sodium/proton antiporters (a) incorporated in BLM are used to selectively transport, object by object, the sodium ions

Table 1. The main results on artificial membrane research (from Ottowa and Tien, 2002) with suggestion on their usefulness for the *in vitro* implementation of P systems

Period	Main results	Significance for P systems
First decade (1961–1970)	Technique for BLM formation; excitability-inducing-material, discrete channel conductance; antigen-antibody, enzyme-substrate interactions photoelectric effects	The model of a single channel; kinetic model of enzymatic reaction based on P systems
Second decade (1971–1980)	Models for the plasma membrane of cells, the nerve membrane, the cristae membrane of mitochondria, the thylakoid membrane of the chloroplast, the visual receptor membrane of the eye, and many others, last but not the least, *a model for the purple membrane of H. halobium*; ion channel reconstruction	The study of discontinuous bioenergetic processes based on electron transfer (respiration, photosynthesis etc.) to write a discrete model for them in order to...
Third decade (1981–1990)	Molecular mechanisms of membrane processes; supported BLMs on metal substrates (s-BLMs); mechanisms of photosynthesis, membrane bioenergetics. Solar energy utilization via semiconductor septum electrochemical photovoltaic cells; electroporation	...construct stable microdevices as hardware components of a P systems-based computer
Fourth decade (1991–2001)	Supported BLMs on hydrogels (sb-BLMs), tethered (t-BLMs). Supported BLM-based sensors and devices. Molecular mechanisms of membrane processes; BLM-based biotechnology and molecular electronics; DNA-BLM interactions	The construction of a P systems-based computer having a P system based software

from the first compartment to the third one. The initial (A) and the final (B) configurations are different, a sequence of transitions occurring in between these two states.

This kind of experiments could lead to the construction of P systems-based computers. In my opinion, these experiments (with antiporters or any other active molecule biologically produced or chemically synthesized, but arranged in an appropriate way within the artificial membrane) could be for P systems what are DNA experiments for DNA computing: *in vitro* use of molecules to calculate. Moreover, there are the following advantages of *in vitro* membrane computing as compared with DNA computing: i) The speed of these processes occurring at membranes (both artificial and natural (seconds, minutes) is much higher as compared with DNA computing experiments, thus leading to a faster computation. ii) The process will occur at a membrane which can function as a microdevice. Furthermore, planar BLMs can now be formed on various substrates with long-term stability, thereby opening the way for basic research and developments

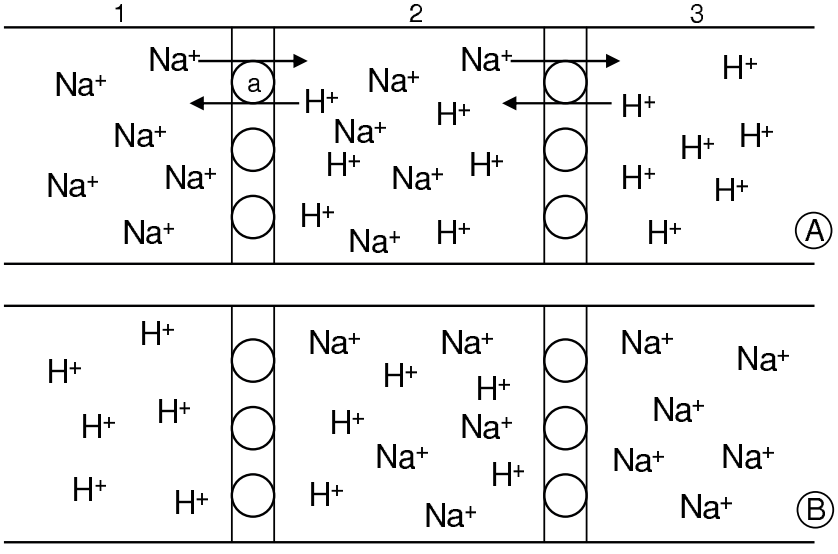


Fig. 1. Schematic representation of an imaginary BLM experiment used to move across an artificial membrane sodium ions from compartment 1 to compartment 3. If the activity of each single antiporter (a) or of a small number of antiporters could be appropriately controlled, the device thus obtained should be useful for the *in vitro* implementation of P systems.

work in biotechnology (Ottowa and Tien, 2002). iii) In many *in vitro* reconstituted systems at/within artificial membranes the output is an electric signal, either current or potential. That could facilitate the transfer of information from one *in vitro* structure to another one, as well as the building up of complex hierarchical structures, using the already acquired knowledge and skills used in the construction of to day computers.

Furthermore, in artificial membranes one could incorporate molecules which function as molecular logic gates such as those active in respiration (Ardelean et al., 2004). Moreover, very recent results show that it is possible to improve the structure of artificial vesicle membranes by coating hollow polyelectrolyte capsules with biological interfaces such as phospholipids membrane and proteins (Moya and Toca-Herrera, 2006), a step toward an artificial cell assembly (Noireaux and Libchaber, 2004). The results in artificial membrane research support the hope that they are appropriate tools for *in vitro* P systems-based experiments.

In conclusion, I claim that P systems are very appropriate to model the function of single objects (cell, channel, enzyme, etc.) and suggest that *in vitro* implementation of P systems could be done by the use of artificial membranes, a step towards computations with artificial membranes.

Acknowledgements

Thanks are due to my main two co-authors, Daniela Besozzi and Matteo Cavaliere, to all the co-authors, S. Aguzzoli, M.H. Garzon, M. Gheorghe, B. Gherla, G. Mauri, C. Manara, V. Mitrana, D. Sburlan, S. Roy, and to all P scientists I had the chance to exchange ideas within a fruitful framework. Special thanks are due to D.G. Mărgineanu and to S. Szedlacseck for helpful discussions and suggestions on thermodynamics in biology and single enzyme activities, respectively. Lorentz Center and PNCDI-CERES (contract 84/2004) are acknowledged for financial support to attend the workshop, and Lucia Dumitru (Head, Centre of Microbiology) for interest in this topic.

References

1. B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, J.D. Watson: *Molecular Biology of the Cell.* 3rd ed., Garland Publishing, New York, 1994.
2. H. Alle, J.R.P. Geiger: Combined analogue and actin potential coding in hippocampus mossy fibres. *Science*, 311 (2006), 1290–1293.
3. I.I Ardelean: The relevance of biomembranes for P systems – general aspects. *Fundamenta Informaticae*, 49, 1-3 (2002), 35–43.
4. I.I. Ardelean: Molecular biology of bacteria and its relevance for P systems. In *Membrane Computing*, LNCS 2597 (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), Springer-Verlag, Berlin, 2003, 1–19.
5. I.I. Ardelean, D. Besozzi, M.H. Garzon, G. Mauri, S. Roy: P system models for mechanosensitive channels. In *Applications of Membrane Computing* (G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds.), Springer-Verlag, Berlin, 2006, 43–81.
6. I.I. Ardelean, D. Besozzi, C. Manara: Aerobic respirations a bio-logic circuit containing molecular logic gates. *Pre-Proc. of Fifth Workshop on Membrane Computing, WMC5* (G. Mauri, Gh. Paŭn, C. Zandron, eds.), Universita' di Milano-Bicocca, June 14-16, 2004, 119–125.
7. I.I. Ardelean, M. Cavaliere: Modelling biological processes by using probabilistic P system software. *Natural Computing*, 2 (2003), 173–197.
8. J.P. Armitage: Bacterial tactic responses. *Adv. Microb. Physiol.,* 41 (1999), 229–289.
9. M.D. Baker, M. Peter, P.M. Wolanin, J.B. Stock: Systems biology of bacterial chemotaxis. *Current Opinion in Microbiology*, 9 (2006), 1–6.
10. D. Besozzi, G. Rozenberg: Extended P systems for the analysis of (trans)membrane protein populations. In *Pre-Proceedings of 7th Workshop on Membrane Computing* (H.J. Hoogeboom, Gh. Păun, G. Rozenberg, eds.), 17-21 July 2006, Leiden Center, 8–10.
11. L. Bianco, F. Fontana, G. Franco, V. Manca: P systems for biological dynamics. In *Applications of Membrane Computing* (G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds.), Springer-Verlag, Berlin, 2006, 81–126.
12. L. Bianco, F. Fontana: Towards a hibrid metabolic algorithm. In *Pre-Proceedings of 7th Workshop on Membrane Computing* (H.J. Hoogeboom, Gh. Păun, G. Rozenberg, eds.), 17-21 July 2006, Leiden Center, 145-158
13. R.P. Blakemore: Magnetotactic bacteria. *Science*, 190 (1975), 377–379.
14. D. Bray: Protein molecules as computational elements in living cells. *Nature*, 376 (1995), 307–312.

15. D. Bray: Bacterial chemotaxis and the question of gain. *PNAS*, 99 (2002), 7–9.
16. B.F. Brehm-Stecher, E.A. Johnson: Single-cell microbiology: tools, technologies, and applications. *Microbiol. Mol. Biol. Rev.*, 68 (2004), 538–559.
17. C.E. Carr, M. Konishi: Axonal delay lines for time measurement in the owl's brainstem. *Proc. Natl. Acad. Sci.*, 85 (1988), 8311–8315.
18. M. Cavaliere: Evolution-communication P systems. In *Membrane Computing* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), LNCS 2597, Springer-Verlag, Berlin, 2003, 134–145.
19. M. Cavaliere I.I. Ardelean: Modelling respiration in bacteria and respiration/photosynthesis interaction in cyanobacteria. In *Applications of Membrane Computing* (G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds.), Springer-Verlag, Berlin, 2006, 129–159.
20. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. In *RGNC Raport* 02/2006, Fenix Editora, Sevillia, 2006, 169–195.
21. H. Chen, M. Ionescu, T.-O. Ishdorj: On the efficiency of spiking neural P systems. In *RGNC Raport* 02/2006, Fenix Editora, Sevillia, 2006, 195–207.
22. H. Chen, M. Ionescu, A. Păun, Gh. Păun, B. Popa: On trace languages generated by spiking neural P systems. In *RGNC Raport* 02/2006, Fenix Editora, Sevillia, 2006, 207–225.
23. H. Chen, T.-O. Ishdorj, Gh. Păun: Computing along the axon. In *RGNC Raport* 02/2006, Fenix Editora, Sevillia, 2006, 225–241.
24. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. In *RGNC Raport* 02/2006, Fenix Editora, Sevillia, 2006, 241–267.
25. G. Ciobanu, M. Pérez-Jiménez, Gh. Păun, eds.: *Applications of Membrane Computing*. Springer-Verlag, Berlin, 2006.
26. A. Ciures, D. Mărgineanu: Thermodynamics in biology: an intruder? *J. Theor. Biol.*, 28, 1 (1970), 147–150.
27. M.H. Garzon, E. Drumwright, R.J. Deaton, D. Renault: Virtual test tubes: A new methodology for computing. In *Proc. 7th Int. Symposium on String Processing and Information Retrieval*, A Corunna, Spain, IEEE Computer Society Press, 2000, 116–121.
28. P. Glandsdorff, I. Prigogine: *Thermodynamics of Structure, Stability and Fluctuations*. Wiley-Interscience, New York, 1971.
29. L.H. Hartwell, J.L. Hopfield, S. Leibler, A.W. Murray: From molecular to modular cell biology. *Nature*, 402 (1999), C47–C52.
30. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodriguez-Paton, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems. In *RGNC Raport* 03/2006, Fenix Editora, Sevillia, 2006, 105–137.
31. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
32. H. Jung: Towards the molecular mechanism of Na/solute symport in prokaryotes. *Biochem. Biophys. Acta*, 1505 (2001), 131–143.
33. H. Kitano: Systems biology – A brief overview. *Science*, 295 (2002), 1662–1664.
34. V. Manca: MP systems approaches to biochemical dynamics: biological rhythms and oscillations. In *Pre-Proc. of Workshop on Membrane Computing* (H.J. Hoogeboom, Gh. Păun, G. Rozenberg, eds.), 17-21 July 2006, Lorentz Center, Leiden, 40–53.
35. V. Manca: Topics and problems in metabolic P systems. In *RGNC Raport* 03/2006, Fenix Editora, Sevillia, 2006, 173–184.

36. A. Martinez-Antonio, J.S. Chandra, H. Salgado, J. Collado-Vides: Internal-sensing machinery directs the activity of the regulatory network in *Escherichia coli*. *Trends in Microbiology*, 1 (2006), 22–27.
37. E. Mayer: *This is Biology*. The Belknap Press of Harvard University Press, 1998.
38. D.G. Mărgineanu: From metaphor to mechanism in membrane biophysics. *Rev. Quest. Scient.*, 172 (2001), 277–292.
39. H.J. Morowitza: *Entropy for Biologists. An Introduction to Thermodynamics*. Academic Press, New York, 1972.
40. S.E. Moya, J.L. Toca-Herrera: From hollow shells to artificial cells: biointerface engineering on polyelectrolyte capsules. *J. Nanosci. Nanotechnol.*, 6 (2006), 1–9.
41. V. Noireaux, A. Libchaber: A vesicle bioreactor as a step toward an artificial cell assembly. *PNAS*, 101 (2004), 17669–17674.
42. A. Ottova, H.T. Tien: The 40th anniversary of bilayer lipid membrane research. *Bioelectrochemistry*, 56 (2002), 171–173.
43. E. Padan, M. Venturi, Y. Gercham, N. Dover: Na/H antiporters. *Biochem. Biophys. Acta*, 1505 (2001), 144–157.
44. B. Palsson: The challenges of in silico biology. *Nature Biotechnology*, 18 (2000), 1147–1150.
45. A. Păun, Gh. Păun: Small universal spiking neural P systems. In *RGNC Raport* 03/2006, Fenix Editora, Sevillia, 2006, 213–235.
46. Gh. Păun: Computing with membranes. *Journal of Computer and Systems Sciences*, 61 (2000), 108–143.
47. Gh. Păun: From cells to computers using membrane (P systems). *BioSystems*, 59 (2001), 139–158.
48. Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
49. M.J. Pérez-Jiménez: P systems based modelling of cellular signalling pathways. *Pre-Proc. of Workshop on Membrane Computing* (H.J. Hoogeboom, Gh. Păun, G. Rozenberg, eds.), 17-21 July 2006, Lorentz Center, Leiden, 54-74.
50. M.J. Pérez-Jiménez, F.J. Romero-Campero: A study of the robustness of the EGFR signalling cascade using continuous membrane systems. LNCS 3651, Springer-Verlag, Berlin, 2005, 268–278.
51. M.H. Saier: Genome archaeology leading to the characterization and classification of transport proteins. *Curr. Op. Microbiol,* 2 (1999), 555–561.
52. I. Segev, E. Schneidman: Axons as computing devices: Basic insights gained from models. *J. Physiol.*, 93 (1999), 263–270.
53. J. Shi, J. Dertouzos, A. Gafni, D. Steel, B.A. Palfey: Single-molecule kinetics reveals signatures of half-sites reactivity in dihydroorotate dehydrogenase A catalysis. *PNAS*, 103 (2006), 5775–5780.
54. H. Szurmant, G.W. Ordal: Diversity in chemotaxis mechanisms among the bacteria and Archaea. *Microbiol. Mol. Biol. Rev.*, 68 (2004), 301–319.
55. A.M. Turing: The chemical basis of morphogenesis. *Phil. Trans. R. Soc. London*, B, 237 (1952), 37–72.

Formalizing Spherical Membrane Structures and Membrane Proteins Populations

Daniela Besozzi[1] and Grzegorz Rozenberg[2]

[1] Università degli Studi di Milano
Dipartimento di Informatica e Comunicazione
Via Comelico 39, 20135 Milano, Italy
besozzi@dico.unimi.it
[2] Leiden Institute of Advanced Computer Science, Leiden University
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
rozenber@liacs.nl

Abstract. We present a formalization of membrane structure by using a parametric 2-dimensional spherical surface, where membrane proteins reside and can move, according to prescribed operations. A more detailed formalization of membrane proteins acting as transporters is also given, thus possibly allowing a global scale analysis of ion flows across a membrane. Several other applications, both biology and computation oriented, are proposed.

1 Introduction

Membrane proteins have many different structures, and perform a whole variety of tasks: they help in regulating the selective permeability of the membrane, the cell signaling and membrane trafficking. Membrane proteins are called *peripheral*, when they are anchored to the internal or external layer of the membrane, and *integral* (or *transmembrane*), when they span the bilayer and face both sides of the membrane. Several important cellular processes, such as the muscle contraction, the transmission of electric impulses in neurons, the response to environmental nutrients, etc., are regulated by different solutes concentrations inside and outside the cell, or by the activation and amplification of specific molecules (called the second messengers) inside the cell. Both phenomena are mediated by transmembrane proteins which, in the case of *transporters* (involved in the active or passive transport of solutes) allow the selective passage of ions (or other small molecules) inwards or outwards, while in the case of membrane *receptors* transduce an external signal towards a downhill chain of reactions inside the cell.

Some basic features of different types of transmembrane proteins were already formalized in the framework of membrane computing ([28,30]), taking into account well known biological notions and the established mechanistic model about the functioning of (single) membrane proteins. For instance, a membrane system model for describing and simulating the activity of mechanosensitive channels of large conductance is given in [6] - these channels are transmembrane proteins gated by mechanical forces exerted on the membrane, and their role is the protection against severe osmotic downshifts in the cell. A similar approach was

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 18–41, 2006.

considered in [9], where a P system model is proposed for the functioning of the sodium-potassium pump, and in [14,26], where the activity of calcium channels and transporters was modelled and stochastically simulated using P systems.

In this paper, we propose to move from the approach of modeling *single* membrane protein to a more global approach of modeling membrane proteins *populations* (MP populations), in order to account for the functioning synergy of *many* membrane proteins (and its effects on the cell life processes). We hope that this modeling will contribute to a global scale analysis of MP populations.

To this aim, we present an extension of the standard notion of membrane in P systems – this extension relies on the use of a 2-dimensional parametric spherical surface instead of a 1-dimensional border. This new approach enables one to consider the spatial distribution of MP populations over the membrane surface. In particular, one can describe such populations using finite sets of circular words which can be naturally associated to the chosen parametrization. Then, the operations over circular words such as insertion and deletion, commutation and shift, formalize various movements of proteins upon the membrane surface, yielding in this way a global dynamic view of the MP populations. In particular, we focus our attention on populations of transporters, by formalizing some biological properties characterizing various types of transport, such as the selectivity, the direction of crossing, and the flux rates. We envisage that this approach may lead to a number of possible applications such as: global ion fluxes ([4]) through the plasma membrane, the dynamical representation of protein movements and clustering, the description of interaction effects between the membrane and the proteins residing in it that may affect the curvature.

The paper is structured as follows. In Section 2 we review the structure of biological membranes, the different types of membrane proteins and their functioning, as well as the classical conceptualization of the membrane architecture (the Singer-Nicholson fluid mosaic model), related to the movements of proteins upon the membrane. In Section 3 we give the definition of the parametric spherical surface and of the set of circular words representing the MP population. In Section 4 we define the operations over circular words that characterize the movements of membrane proteins upon the surface. Then, in Section 5, we reduce the level of abstraction in the description of MP populations, by giving a more detailed representation of transport proteins. Finally, in Section 6 we present in some detail some possible applications of the extended membrane framework for the analysis of both biological and computationally oriented topics.

2 Biological Membranes and Membrane Proteins

In this section we revise some notions concerning cellular membranes, such as its lipid and protein constituents, and the standard conceptualizations of the membrane structure and fluidity. Then, we describe three major classes of membrane proteins that are involved in the transport of solutes across the membrane. For more details and examples we refer the reader to standard books in Molecular Cell Biology such as [23,3].

Cell's shape. For some types of cells (e.g. neural or vegetal cells), the shape is a nonvariable characteristic. Other cells (e.g. migrating blood cells) are subject to shape variation which, besides possibly depending also on the cytoskeleton, is mostly due to environmental actions of mechanistic type (the pressure exerted by neighboring cells, or by the surrounding biological liquid) and to their functionality in the tissue or organism. Whenever the cells that do not possess a specific structured shape are artificially isolated from the tissue they belong to, they gradually resume a spherical form, according to physical laws of surface tension. Similarly, when spherical shaped cells are put close to each other, they assume a polyhedral shape with the number of faces corresponding to the number of cells in mutual contact.

Constituents of biological membranes. The primary constituents of cellular membranes (whether they are plasma or internal membranes) are amphipathic molecules (i.e., molecules with one hydrophobic and one hydrophilic part) called *phospholipids*. There are several classes of phospholipids, such as phosphoglycerides, sphyngolipids, cholesterol, steroids, glycolipids, ... – they differ with respect to the molecular composition of the hydro-phobic/philic parts. Due to the amphipatic property of phospholipids, biological membranes are composed of sheetlike bilayers (with two external hydrophilic polar parts and and internal hydrophobic apolar part) that spontaneously form closed structures that separate two aqueous compartments. Globular proteins are also associated to the bilayer in different ways. *Peripheral* proteins are (more or less weakly) bound to the hydrophilic part of phospholipids, or dipped into the bilayer – but not interacting with its hydrophobic core – on both sides of the membrane, facing either the cytoplasm or the environment ([13]). *Integral* (or transmembrane) proteins have one or more domains (α helices or multiple β strands)[1] that span the whole depth of the bilayer. Integral proteins have a very specific orientation (i.e., an asymmetric structure) with respect to the two sides of the membrane, conferring different properties on the two faces; this fact is also reflected by the different molecular compositions in the two compartments separated by the membrane. The orientation of an integral protein is established during its biosynthesis and its insertion into the endoplasmic reticulum membrane, by means of complex structures called translocons (see, e.g., [23]).

The fluid mosaic model. The fluidity and dynamic mobility of a cellular membrane are determined by its specific lipid and protein composition, which can differ from one layer to another in a membrane, from membrane to membrane (with different membranes surrounding different organelles in a cell), and from cell to cell. Due to thermal motions, phospholipids and glycolipids can freely rotate around their longitudinal axis, and also diffuse laterally within the layer where they reside. Only occasionally, they can flip-flop from one layer to the other one (a motion that needs to be catalyzed by proteins called flippases). Many integral proteins are also freely mobile; immobile proteins are those that are

[1] We emphasize that here we are not concerned with a molecular-scale description of the aminoacidic composition or the conformations of proteins, but only with their functioning mechanisms and activation conditions (see also Sections 5 and 6.1).

permanently attached to the cytoskeletal elements inside the cytoplasm. The first conceptualization of the membrane structure, known as the *fluid mosaic model*, was proposed in the seventies by Singer and Nicholson ([35,34]). According to this model, the cellular membrane is a 2-dimensional "sea of lipids" where lipid molecules, as well as (monomeric and low abundant) integral proteins, freely float unencumbered. Criticisms of this model, and new insights for a (plausibly more) realistic paradigm of membrane architecture and mobility, will be further discussed in Section 6.4.

Transport proteins. The plasma membrane delimits the cell, and acts as a selective permeable barrier between the cytoplasm and the exoplasm, allowing a bidirectional passage of molecules. Its functions include the inflow of nutrients and the outflow of waste solutes, the maintenance of proper ionic composition and pH, the transduction of external signals, the exchange of metabolites between adjacent cells in a tissue, the interactions with the extracellular matrix. Most of these functions are carried out by specific transmembrane proteins. Here, we focus on the activity of the integral proteins involved in the selective transport of ions and solutes – refereeing to them as the *transport proteins*. In different cell types, the transport proteins residing within the plasma membrane, or within various organelle membranes, can vary considerably, both in types and concentrations, thus allowing only certain ions or molecules to cross the membrane.

Transport proteins create the (pH, ions concentration, potential) gradients across the membrane by moving ions and solutes inwards and outwards. At the same time, these gradients are the primary "forms of energy" which can then be used by other transport proteins for the accumulation or the exclusion of other solutes. In this paper, we will consider three major classes of transport proteins: *ATP-powered pumps*, *cotransporters* and *channels*. This classification will be used to distinguish between different transport mechanisms. However some transport proteins may utilize more than one transport mechanism - such cases will not be considered in this paper. Also, we will not explicitly discuss the functioning of other classes of transport proteins, such as, e.g., ABC superfamily proteins ([16]) and porin families.

All transport proteins belonging to the three classes listed above exhibit a high specificity for the transported substances, while they differ with respect to the rate of transport, which is due to different mechanisms of action. Specificity or selectivity means that each transport protein type is able to bind and move a single species, or a single group of closely related molecules with which it has a high (chemical) affinity ([19]).

ATP-powered pumps, or simply pumps, use the energy released by ATP hydrolysis to move ions or small molecules against (or "uphill") an electrochemical gradient (the process is also known as active transport). The coupling between the uphill transport and the hydrolysis of ATP is then energetically favorable. Anyway, the overall mechanism of action consistently slows down the transport rate of pumps to only about 1-10^3 molecules per second.

Cotransporters simultaneously bind only one or few solutes – then a change in the protein conformation allows the transport of bound molecules across the

membrane. Due to the necessary conformational change, cotransporters move about 10^2-10^4 molecules per second. Among cotransporters, *uniporters* move one molecule at a time down its concentration gradient (this process is also known as facilitated transport), while *symporters* and *antiporters* couple the passage of one type of molecule against its concentration gradient with the passage of a different type of molecule down its concentration gradient.

In contrast to the other two classes, *channels* simultaneously transport multiple water molecules or specific types of ions, in a single file, down their concentration or electric potential gradients at a very rapid rate (about 10^7-10^8 molecules per second). Some channels are usually open within the membrane, e.g. the potassium-specific channel, others are usually closed, opening only in response to specific signals.

All symporters and some antiporters move ions together with small molecules, whereas ion pumps and ion channels transport only ions. The rate of ions movement across a membrane is influenced by the external and internal concentrations, as well as by the electric potential existing across the membrane. The ionic gradients of the principal cellular ions (Na^+, K^+, Ca^{2+}) are generated and maintained by ATP-powered pumps. In animal cells these gradients, together with the selective transport of ions through channels, determine an electrochemical potential of around -70mV – with the cytosolic face of the plasma membrane always negative with respect to the exoplasmic face. We refer to Section 6.1 for a description and a formalization of concentration and potential gradients.

3 The Parametric Spherical Membrane and Membrane Proteins Populations

The concept of membrane in P systems ([28]) consists of a (1-dimensional) border – implicitly assumed to correspond to a closed surface in a 3-dimensional space – which identifies and separates two regions. Objects occurring in one region can cross the membrane, by the application of appropriate rules, and thus be placed in the outer region or inside an inner region, if any. Membranes can also be used to represent the lipid bilayer, where objects can reside ([9]), or where specific operations can occur ([27]). Labels attached to the membrane can have various meanings: numeric identifiers, electric charges ([29]), multisets of objects, etc. In the last case, the objects placed *on* the membrane are usually interpreted as proteins associated with the membrane itself, allowing several operations acting directly upon the membrane ([12]).

However, when attempting to describe MP populations, one has to face some limitations of the above concept of membrane. For instance, there is no obvious or easy method to represent a spatial distribution of objects within/upon the membrane. Indeed, one could associate with the membrane a *string* instead of a multiset; in this way, it would be possible to characterize, for each symbol in the string, which are its left and right adjacent symbols. Anyway, the fact that the membrane is represented by a 1-dimensional border only allows for a planar representation of a cellular membrane (like cutting a section in the cell surface).

But the implicit concept of surface would be lost: even by associating a string to the membrane, it still would not be possible to define a notion of neighborhood along every directions on the membrane surface.

For a formal description of a MP population – where the spatial distribution of proteins is important – it is thus more appropriate to consider a 2-dimensional surface which is topologically equivalent (homeomorphic) to the sphere $S^2 \subseteq \mathbb{R}^3$, see [21]. As a first approximation we choose to use the shape of the sphere, though cells can have very different forms, ranging from the biconcave round shape of red blood cells to the highly branched structure of multipolar neurons.

Let Σ be the spherical surface, and consider the canonical discrete parameterization of Σ given by two ordered sets, $\mathcal{P} = (p_1, \ldots, p_r)$ and $\mathcal{M} = (m_1, \ldots, m_s)$, whose elements are called *parallels* and *meridians*, respectively. This parameterization gives rise to the set P of rs intersection points of parallels with meridians, plus two additional, north and south, "polar points" p_N, p_S, which correspond to the sites where all meridians intersect. We distinguish one meridian, viz., $m_1 \in M$, as the "Greenwich" meridian, and starting from m_1 we move counterclockwise on the sphere; similarly, we distinguish one parallel, viz., p_1, as the "northern" parallel which is adjacent (closest to) the north polar point. The meridian m_1 will be used to identify the first symbol in the circular strings corresponding to the parallels, as explained below.

We assume that the canonical parametrization of Σ is regular, meaning that, for any pair of adjacent meridians m_i, m_{i+1}, $i = 1, \ldots, s$, their mutual distance – taken along any common fixed parallel – is the same (equal to $d_{\mathcal{M}}$), and, similarly, for any pair of adjacent parallels p_j, p_{j+1}, $j = 1, \ldots, r$, their mutual distance is the same (equal to $d_{\mathcal{P}}$). This is illustrated in Fig. 1. In this way the values of r, s determine the granularity of the parametrization – obviously, for $r, s \to \infty$, we obtain a continuous surface. In the following, we will also refer to this spherical parametric surface as the $(\mathcal{P}, \mathcal{M})$-membrane.

The parametrization of the spherical surface allows us to formalize MP populations residing in membranes. The resulting notion of a *configuration* of MP population over Σ has to satisfy two assumptions:

1. *placement points*: the intersection points of meridians with parallels are the only locations where any membrane protein can reside upon Σ;
2. *self-avoidance sharing*: any intersection point can be occupied by at most one membrane protein.

A way to achieve such a formalization is to use circular words.

First of all we formalize the spherical grid of intersection points. To this aim we consider a finite and ordered set of circular words $\pi_1, \ldots, \pi_r$ each of length s, and two words π_0, π_{r+1} of length 1. Each π_i, $1 \leq i \leq r$, corresponds to the intersection points of meridians with the parallel p_i, while π_0, π_{r+1} correspond to polar points p_N and p_S, respectively (in fact $\pi_0 = p_N$ and $\pi_{r+1} = p_S$).

Each circular word π_i, $i = 1, \ldots, r$, will be written in the standard form

$$\pi_i = \pi_i^1 \pi_i^2 \cdots \pi_i^s,$$

where $\pi_i^j \in P$.

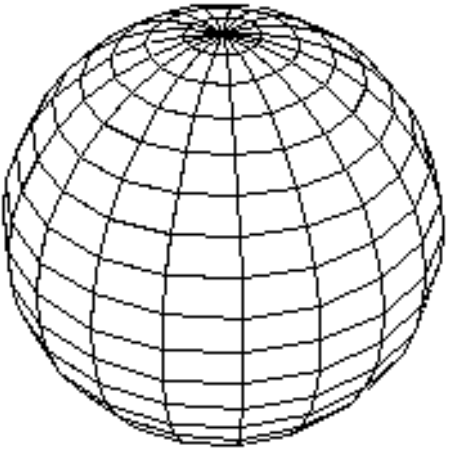


Fig. 1. The parametric regular spherical membrane

The ordered set of circular words $(\pi_0, \pi_1, \ldots, \pi_r, \pi_{r+1})$, called the *scaffold*, represents the spherical grid of intersection points and hence formalizes the structure of the parametric spherical membrane.

Since each π_i, $1 \leq i \leq r$, is a circular word, the counting while moving (along a parallel) to the right or to the left is done modulo s. Thus, $s + 1 = 1$ and $1 - 1 = s$ and so, moving from π_i^s to the right leads to π_i^1, and moving from π_i^1 to the left leads to π_i^s. This yields the following definition of neighborhood.

Definition 1. Let $\sigma = (\pi_0, \pi_1, \ldots, \pi_r, \pi_{r+1})$ be the scaffold.

(1) Let $2 \leq i \leq r - 1$ and consider π_i^j, $1 \leq j \leq s$. The *direct neighborhood of* π_i^j (*in* σ) is the set $\mathcal{N}_\sigma(\pi_i^j) = \{\pi_{i-1}^{j-1}, \pi_{i-1}^{j}, \pi_{i-1}^{j+1}, \pi_i^{j-1}, \pi_i^{j+1}, \pi_{i+1}^{j-1}, \pi_{i+1}^{j}, \pi_{i+1}^{j+1}\}$.
(2) Let $i = 1$ and consider π_1^j, $1 \leq j \leq s$. The *direct neighborhood of* π_1^j (*in* σ) is the set $\mathcal{N}_\sigma(\pi_1^j) = \{p_N, \pi_1^{j-1}, \pi_1^{j+1}, \pi_2^{j-1}, \pi_2^{j}, \pi_2^{j+1}\}$.
(3) Let $i = r$ and consider π_r^j, $1 \leq j \leq s$. The *direct neighborhood of* π_r^j (*in* σ) is the set $\mathcal{N}_\sigma(\pi_r^j) = \{\pi_{r-1}^{j-1}, \pi_{r-1}^{j}, \pi_{r-1}^{j+1}, \pi_r^{j-1}, \pi_r^{j+1}, p_S\}$.
(4) The *direct neighborhood of* the north polar point p_N (*in* σ) is the set $\mathcal{N}_\sigma(p_N) = \{\pi_1^1, \ldots, \pi_1^s\}$.
(5) The *direct neighborhood of* the south polar point p_S (*in* σ) is the set $\mathcal{N}_\sigma(p_S) = \{\pi_r^1, \ldots, \pi_r^s\}$.

In cases (1), (2), and (3) we refer to π_i^{j-1} as the *left neighbor* (*in* σ) of π_i^j, and to π_i^{j+1} as the *right neighbor* (*in* σ) of π_i^j.

Now we move to formalize the placement of proteins in the intersection points of the spherical grid (scaffold).

Let $V = \{A_1, \ldots, A_n\}$ be the alphabet of n different types of membrane proteins, $V_* = V \cup \{*\}$ where $*$ is a special symbol, $* \notin V$, and let $I = \{1, \ldots, r\}$ and $J = \{1, \ldots, s\}$. We will consider an ordered set of circular words $w_1, \ldots, w_r$ –

each of length s, and two words w_0, w_{r+1} of length 1. Each w_i, $i \in I$, corresponds to a placement of proteins along the parallel p_i, while w_0, w_{r+1} correspond to a placement of proteins on the polar points p_N and p_S, respectively.

Each circular word w_i, $i \in I$, will be written in the standard form

$$w_i = A_i^1 A_i^2 \cdots A_i^s,$$

where $A_i^j \in V_*, j \in J$. The interpretation for (the intuition behind) the occurrences of symbols A_i^j in w_i is as follows:

1. the superscript j identifies the intersection point of p_i with m_j,
2. $A_i^j = A_k$, for some $1 \leq k \leq n$, indicates that a protein of type A_k is located at the intersection of p_i with m_j,
3. $A_i^j = *$ indicates that no protein is located at the intersection of p_i with m_j.

We will also use the notation $w_0 = B_0$ and $w_{r+1} = B_{r+1}$, where $B_0, B_{r+1} \in V_*$.

We are now ready to define a central notion of a configuration.

Definition 2. Let Σ be a $(\mathcal{P}, \mathcal{M})$-membrane. A *configuration* γ of Σ is a sequence $\gamma = (w_0, w_1, \ldots, w_r, w_{r+1})$ of circular words over V_*, where each of $w_1, \ldots, w_r$ is of length s and w_0, w_{r+1} are of length 1.

The intuition behind a configuration γ is as follows: the sequence $(w_0, w_1, \ldots, w_r, w_{r+1})$ describes the placement of proteins on all the intersection points of the scaffold $(\pi_0, \pi_1, \ldots, \pi_r, \pi_{r+1})$, where $*$ indicates that there is no protein placed at a given intersection point. Note that this definition of a configuration naturally satisfies the requirements of placement points and self-avoidance sharing (stated in the initial part of this section).

4 Operations

In this section we define operations over circular words which are used to simulate various movements of membrane proteins upon the membrane surface. In what follows, let $\gamma = (w_0, w_1, \ldots, w_r, w_{r+1})$ be a configuration of Σ.

Definition 3. A *commutation* (*in* γ) is a function $com_\gamma : I \times J \to V_*^s$ such that $com_\gamma(i,j)$ is defined if $A_i^j, A_i^{j+1} \neq *$, and when defined $com_\gamma(i,j) = (A_i^1)' \cdots (A_i^s)'$, where $(A_i^j)' = A_i^{j+1}$, $(A_i^{j+1})' = A_i^j$ and $(A_i^h)' = A_i^h$ for all $h \in J - \{j, j+1\}$.

Thus $com_\gamma(i,j)$ exchanges the positions of two adjacent proteins (A_i^j and A_i^{j+1}) residing on the parallel p_i.

Definition 4. A *left shift* (*in* γ) is a function $lsh_\gamma : I \times J \to V_*^s$ such that $lsh_\gamma(i,j)$ is defined if $A_i^j \neq *$ and $A_i^{j-1} = *$, and when defined $lsh_\gamma(i,j) = (A_i^1)' \cdots (A_i^s)'$, where $(A_i^{j-1})' = A_i^j$, $(A_i^j)' = *$ and $(A_i^h)' = A_i^h$ for all $h \in J - \{j, j-1\}$.

Thus $lsh_\gamma(i,j)$ moves protein A_i^j to its *free* left neighbor position.

Definition 5. A *right shift* (*in* γ) is a function $rsh_\gamma : I \times J \to V_*^s$ such that $rsh_\gamma(i,j)$ is defined if $A_i^j \neq *$ and $A_i^{j+1} = *$, and when defined $rsh_\gamma(i,j) = (A_i^1)' \cdots (A_i^s)'$, where $(A_i^j)' = *$, $(A_i^{j+1})' = A_i^j$ and $(A_i^h)' = A_i^h$ for all $h \in J - \{j, j+1\}$.

Analogously to the left shift, $rsh_\gamma(i,j)$ moves protein A_i^j to its *free* right neighbor position.

Definition 6. A *downwards exchange* (*in* γ) is a function $dch_\gamma : (I \cup \{0\}) \times J \to (V_*^s \times V_*^s) \cup (V_*^s \times V_*) \cup (V_* \times V_*^s)$ such that $dch_\gamma(i,j)$ is defined if $A_i^j \neq *$, and when defined then:

1. for $1 \leq i < r$, $dch_\gamma(i,j) = ((A_i^1)' \cdots (A_i^s)', (A_{i+1}^1)' \cdots (A_{i+1}^s)')$, where for some $h \in \{j-1, j, j+1\}$, $(A_i^j)' = A_{i+1}^h$, $(A_{i+1}^h)' = A_i^j$, $(A_{i+1}^t)' = A_{i+1}^t$ for all $t \neq h$, and $(A_i^t)' = A_i^t$ for all $t \neq j$;
2. $dch_\gamma(r,j) = ((A_r^1)' \cdots (A_r^s)', (B_{r+1})')$, where $(A_i^j)' = B_{r+1}$, and $(B_{r+1})' = A_i^j$;
3. $dch_\gamma(0,j) = ((B_0)', (A_1^1)' \cdots (A_1^s)')$, where $(B_0)' = A_1^j$, and $(A_1^j)' = B_0$.

Thus $dch_\gamma(i,j)$ for $1 \leq i < r$ exchanges A_i^j with one of $A_{i+1}^{j-1}, A_{i+1}^j, A_{i+1}^{j+1}$. For $i = r$, $dch_\gamma(r,j)$ exchanges A_r^j with B_{r+1}. For $i = 0$, $dch_\gamma(0,j)$ exchanges B_0 with A_1^j.

Definition 7. An *upwards exchange* (*in* γ) is a function $uch_\gamma : (I \cup \{r+1\}) \times J \to (V_*^s \times V_*^s) \cup (V_*^s \times V_*) \cup (V_* \times V_*^s)$. Analogous to the downwards exchange, $uch_\gamma(i,j)$ is defined if $A_i^j \neq *$, and when defined then:

1. for $1 < i \leq r$, $uch_\gamma(i,j) = ((A_i^1)' \cdots (A_i^s)', (A_{i-1}^1)' \cdots (A_{i-1}^s)')$, where for some $h \in \{j-1, j, j+1\}$, $(A_i^j)' = A_{i-1}^h$, $(A_{i-1}^h)' = A_i^j$, $(A_{i-1}^t)' = A_{i-1}^t$ for all $t \neq h$, and $(A_i^t)' = A_i^t$ for all $t \neq j$;
2. $uch_\gamma(1,j) = ((A_1^1)' \cdots (A_1^s)', (B_0)')$, where $(A_1^j)' = B_0$, and $(B_0)' = A_1^j$;
3. $uch_\gamma(r+1,j) = ((B_{r+1})', (A_r^1)' \cdots (A_r^s)')$, where $(B_{r+1})' = A_r^j$ and $(A_r^j)' = B_{r+1}$.

Thus, analogously to downwards exchange, $uch_\gamma(i,j)$, for $1 < i \leq r$, exchanges A_i^j with one of $A_{i-1}^{j-1}, A_{i-1}^j, A_{i-1}^{j+1}$. For $i = 1$, $uch_\gamma(1,j)$ exchanges A_1^j with B_0. For $i = r+1$, $dch_\gamma(r+1,j)$ exchanges B_{r+1} with A_r^j.

In Figure 2 we graphically represent the set of direct neighbors of protein A_i^j placed on p_i, for some $1 < i < r$. The direct neighbors are the only intersection points to which protein A_i^j can move to by the application of one commutation, one shift or one exchange operation.

Besides the operations that describe the possible movements of proteins within a $(\mathcal{P}, \mathcal{M})$-membrane, we also define two operations which allow one to add or remove proteins from Σ.

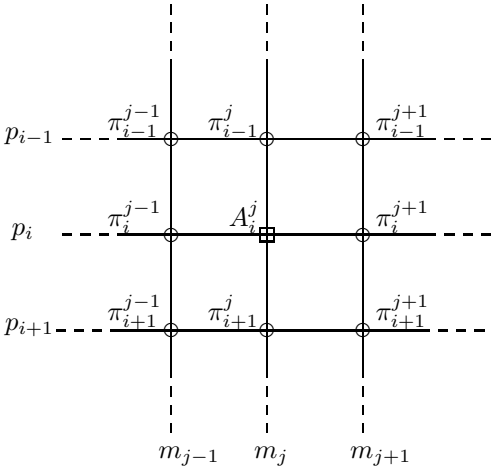


Fig. 2. The set of direct neighbors (circle intersection points) of protein A_i^j (square intersection point)

Definition 8. A *deletion* (*in* γ) is a function $del_\gamma : (I \cup \{0, r+1\}) \times J \to V_*^s$ such that $del_\gamma(i,j)$ is defined if $A_i^j \neq *$, and when defined $del_\gamma(i,j) = (A_i^1)' \cdots (A_i^s)'$, where $(A_i^j)' = *$ and $(A_i^h)' = A_i^h$ for all $h \in J - \{j\}$.

Thus $del_\gamma(i,j)$ removes the protein A_i^j from configuration γ.

Definition 9. An *insertion* (*in* γ) is a function $ins_\gamma : (I \cup \{0, r+1\}) \times J \times V \to V_*^s$ such that $ins_\gamma(i,j,A)$ is defined if $A_i^j = *$, and when defined $ins_\gamma(i,j,A) = (A_i^1)' \cdots (A_i^s)'$, where $(A_i^j)' = A$ and $(A_i^h)' = A_i^h$ for all $h \in J - \{j\}$.

Thus $ins_\gamma(i,j,A)$ inserts protein A into the vacant position A_i^j in configuration γ.

Note that the application of commutation, shift and exchange operations produces a different placement over Σ of the proteins already residing in it. On the other hand, insertion and deletion operations cause the current MP population (namely, the number and possibly the types of proteins over the surface) to change. Indeed, the molecular composition of a cellular membrane is not fixed during all cell's life, but it is a dynamic constituent which changes according to proteins lifetime and vesicular trafficking.

For a configuration γ, we will use OP_γ to denote the set of operations that we have defined above, viz., commutation, left and right shift, downwards and upwards exchange, insertion and deletion.

Two assumptions can be done concerning the MP population initially present upon the parametric spherical surface. On the one hand, we can assume that the proteins are randomly placed over the membrane, provided the obtained configuration satisfies the requirements of placement point and self-avoidance sharing. On the other hand, we can require that the placement of proteins correspond to

some biological restriction, e.g., the clustering of specific types of proteins, the presence of lipid rafts, the description of membrane patchiness (see Section 6 for a discussion of these topics).

In both cases, starting from an initial configuration γ over Σ, it is possible to describe the *evolution* of the MP population. For instance, for the analysis of the real processes discussed in Section 6 – such as the mechanisms generating the curvature of the membrane, the movement of proteins according to membrane "fence" models, etc. – specific constants can also be associated to each operation in OP_γ. This will allow, e.g., to follow the dynamics of the MP population in a stochastic manner.

To this aim, in Section 5 we will associate the notion of *functionality* to the transmembrane proteins acting as transporters. Then, in Section 6.1 we will describe how to use this feature for the global analysis of the ion flows across a membrane.

5 A Formal Description of Transport Proteins

In this section we formalize the functionality of transmembrane proteins involved in the passage of solutes across the cellular membrane.

Let Σ be a $(\mathcal{P}, \mathcal{M})$-membrane as defined in Section 3 and $\mathcal{O}$ an alphabet of objects. Let us denote by P_{out} and P_{in} the multisets over $\mathcal{O}$ occurring, respectively, in the region outside Σ (the external region) and in the region delimited by Σ (the internal region).

Definition 10. For each $A \in V$, the *functionality* of A, denoted by $f(A)$, is a subset of the cartesian product $\mathcal{O} \times \{\uparrow, \downarrow, \updownarrow\} \times \mathbb{N}^+$ such that, for all $(x, y, z) \in f(A)$, if $(x, y', z') \in f(A)$ then $y' = y$ and $z' = z$.

The intuition behind this definition is as follows: $(x, y, n) \in f(A)$ means that A is selective for object $x \in \mathcal{O}$ and simultaneously transports n copies of x according to direction y: for $y = \uparrow$, x is transported to the external region, for $y = \downarrow$, x is transported to the internal region, and for $y = \updownarrow$, x may be transported to either external or to internal regions.

Example 1. Let $V = \{A, B, C\}$ and $\mathcal{O} = \{a, b, c, d, e\}$. Let the functionality of proteins A, B and C be defined as follows:

$$f(A) = (a, \updownarrow, 1)$$
$$f(B) = \{(b, \downarrow, 3), (c, \uparrow, 2)\}$$
$$f(C) = \{(b, \downarrow, 1), (d, \downarrow, 1), (e, \uparrow, 1)\}$$

This means that:

- any protein of type A is selective for object $a \in \mathcal{O}$ only, it moves exactly 1 copy of a either from the external to the internal region, or the other way around;

- any protein of type B is selective for objects $b, c \in \mathcal{O}$, it simultaneously moves exactly 3 copies of b from the external to the internal region and exactly 2 copies of c from the internal to the external region;
- any protein of type C is selective for objects $b, d, e \in \mathcal{O}$, it simultaneously moves exactly 1 copy of b and 1 copy of d from the external to the internal region, and exactly 1 copy of e from the internal to the external region.

Remark 1. Consider the abstract protein types described in Example 1, and let a, b, c, d, e correspond to glucose, Na^+, K^+, HCO_3^+, Cl^-, respectively. Then, protein A could correspond to *GLUT1 uniporter* in mammalian cells, which transports one molecule of glucose either from the external to the internal region, or the other way around, according to the glucose concentration gradient across the membrane (it always performs a "downhill" transport). Protein B could correspond to *Na^+-K^+ pump*, which exchanges three extracellular Na^+ ions with two intracellular K^+ ions (by consuming one ATP molecule). Protein C could correspond to *Na^+-HCO_3^+/Cl^- antiporter*, which exploits the downhill concentration gradient of Na^+ to simultaneously move one Na^+ and one HCO_3^+ from the external to the internal region, and one Cl^- in the opposite direction, against its concentration gradient. See, e.g., [23] for more details about the mentioned transport proteins.

Remark 2. If $A \in V$ is a protein type corresponding to a channel, then the meaning of the value n in the definition of the functionality $f(A)$ has to be relaxed. Indeed, protein channels do not transport a *fixed* number of molecules – that is, they are not characterized by a very specific stoichiometry. To the contrary, many molecules can simultaneously cross a channel whenever it is open: a current, or *flow*, can be experimentally determined for the functioning of (a single occurrence of) each channel type ([36]). We explain here how to associate an appropriate value n to protein channel types. Let I be the known current of channel A, let Z be the electric charge of each transported molecule, and $q = 1.6021773 \cdot 10^{-19}$ C the value of the elementary charge (see, e.g., [7]). Then, the *average number* of molecules that are simultaneously transported by a channel A in one second is given by the formula $n = I/(Zq)$. As an example, we derive the average number of Ca^{2+} ions which are transported in one second by the ryanodin receptor channel (RyR) ([18]). Since $I_{RyR} = 0.3$ pA and $Z_{Ca^{2+}} = 2$, then $n_{RyR} \simeq 9.4 \cdot 10^5$ ions/sec (see also [26]).

We emphasize that up to now we have only considered the *functioning* of transport proteins. In Section 6.1 we propose a formal description of some biological conditions, e.g. energy consumption, concentration and potential gradients, that trigger the *activation* of such proteins.

We are ready now to formulate a method of object transport based on MP populations. A *membrane population transport scheme*, abbreviated MPT scheme, is a four tuple $S = (V, \Sigma, F_V, \mathcal{O})$ such that:

- V is an alphabet of protein types;
- Σ is a $(\mathcal{P}, \mathcal{M})$-membrane over V;

- $F_V = \{f(A) \mid A \in V\}$ is the set of functionalities for protein types in V;
- $\mathcal{O}$ is an alphabet of *objects*.

A MPT scheme S defines a scheme of transporting molecules (objects) through a $(\mathcal{P}, \mathcal{M})$-membrane. It provides the specification (F_V) of how molecules (objects from $\mathcal{O}$) are transported by various protein types (from V) through any distribution of proteins within the given $(\mathcal{P}, \mathcal{M})$-membrane Σ. A distribution of proteins within Σ is given by a configuration of Σ. For a given configuration γ one can determine both the transport of molecules through Σ in this configuration γ, as well as the dynamic change of γ through the operations in OP_γ defined in Section 4.

The behavior of S, hence the transport of objects through Σ together with the dynamic changes of Σ, can be as usual formalized through transitions between the instantaneous descriptions of S. Such a formalization could proceed as follows.

An *instantaneous description of* S is a triplet $C = (\gamma, P_{in}, P_{out})$ where γ is a configuration of Σ and P_{in}, P_{out} are multisets over $\mathcal{O}$. The underlying intuition is that γ represents the distribution of proteins within Σ, and P_{in}, P_{out} represent multisets of objects present in the inside and outside regions of Σ at a moment of time captured (formalized) by C.

Then the *transition from* C *to* $C' = (\gamma', P'_{in}, P'_{out})$ *in* S, denoted $C \vdash_S C'$, can take place if:

1. γ' is obtained from γ by the parallel application of operations from OP_γ, with the assumption that each coordinate $A_i^j \in V_*$ in Σ is the subject of at most one operation in OP_γ;
2. P'_{in}, P'_{out} are obtained by the parallel transport of objects from P_{in} and P_{out} according to the functionality of proteins in γ.

The *evolution of* S *beginning with* C is a sequence $C_0, C_1, \ldots, C_n$, $n \geq 1$, of instantaneous descriptions such that $C_0 = C$ and, for each $0 \leq i \leq n-1$, $C_i \vdash_S C_{i+1}$.

Then, by fixing the initial multisets of objects in the inside and the outside regions of Σ, one obtains a membrane population transport system.

Thus a *membrane population transport system*, abbreviated MTP system, is a triplet $T = (S, Q_{in}, Q_{out})$ such that S is a MTP scheme and Q_{in}, Q_{out} are multisets over the alphabet of objects of S, called the *internal* and *external* multisets of T, respectively. An evolution of T is an evolution of S beginning with an instantaneous description $C = (\gamma, P_{in}, P_{out})$ such that $P_{in} = Q_{in}$ and $P_{out} = Q_{out}$.

6 Discussion

In this section we discuss some possible applications of the parametric spherical membrane and MP populations. Biological and formal aspects of possible research lines are explained; detailed description and specific analysis concerning the presented topics will be further presented in forthcoming papers.

We begin by discussing biologically oriented topics such as global ions flows at the whole plasma membrane, new conceptualizations of the membrane architecture, the presence of membrane microdomains and protein clusters, and some membrane curvature generating mechanisms.

Then, we propose some applications of our framework for the analysis of computation oriented topics.

6.1 Ion Flows

In Section 5 we have defined the functionality $f(A)$ of a protein type $A \in V$ which determines the selective transport of objects across the membrane, their multiplicity and crossing direction. In order to achieve a complete characterization of protein transport (e.g., for the purpose of analysis of the ion flows across a membrane) it is also important to know which proteins – in any evolution step – can be actually functioning, according to the current environmental conditions. To this aim, we formalize the notions related to some biological conditions that can trigger the activation of a transport protein. By *activation* we mean the (conformation) change of an individual transport protein into its functional state. The biological conditions that we consider here are the concentration gradient and potential gradient across the membrane, as well as the availability of energy molecules. Unless otherwise specified, we assume that the distribution of ions – inside and outside the membrane – is homogeneous. Moreover, we will only consider the contribution to membrane gradients given by the transported ions and molecules, viz., the objects from the alphabet $\mathcal{O}$.

We begin by the evaluation of the concentration and potential gradients across a membrane. Let $C = (\gamma, P_{in}, P_{out})$ be an instantaneous description of S. Let $P_{in}(a), P_{out}(a)$ denote the occurrences of object $a \in \mathcal{O}$ in the multisets P_{in} and P_{out}, respectively[2].

Definition 11. The *concentration gradient* across a membrane – with respect to object $a \in \mathcal{O}$ – is defined by $\Delta Conc_a = P_{out}(a) - P_{in}(a)$.

The activation of a transport protein A can be influenced by the concentration gradients corresponding to the objects that are selectively transported by A. Hence, given the functionality $f(A)$ of protein A, for ion flux analysis one should account for the set $\Delta Conc_{f(A)} = \{\Delta Conc_x \mid (x, y, n) \in f(A)\} \subseteq \mathbb{N}^\alpha$, where α is the number of distinct symbols $x \in \mathcal{O}$ such that $(x, y, n) \in f(A)$.

To define the potential (or voltage) gradient across a membrane, we first need to associate a charge to each object $a \in \mathcal{O}$. We use the notation $a^{k_a c_a}$ to say that the *charged object* a has k_a units of (positive or negative) charge c_a, for some $k_a \in \mathbb{N}^+$ and for $c_a \in \{+, -\}$.

Example 2. Let $a, b \in \mathcal{O}$ correspond to ions Ca^{2+} and Cl^-, respectively. Then, $c_a = +, k_a = 2$ and $c_b = -, k_b = 1$.

[2] Discrete multiplicities of ions can be derived from real (molar) concentration values by considering also the volume values of the involved regions.

Definition 12. The *voltage gradient* across a membrane is defined by $\Delta Volt = \sum_{a \in \mathcal{O}} k_a c_a \cdot \Delta Conc_a$.

Thus, the voltage gradient is evaluated by considering the difference between the sum of all external charges and the sum of all internal charges. In fact, $\Delta Volt = \sum_{a \in \mathcal{O}} k_a c_a \cdot \Delta Conc_a = \sum_{a \in \mathcal{O}} k_a c_a \cdot (P_{out}(a) - P_{in}(a)) = \sum_{a \in \mathcal{O}} (k_a c_a \cdot P_{out}(a)) - \sum_{a \in \mathcal{O}} (k_a c_a \cdot P_{in}(a))$.

The factor c_a in the definition of $\Delta Volt_a$ is used to comply with the convention that, for positively charged ions, the membrane potential is the difference between the potential on the external face and the potential on the internal face, while for negatively charged ions it is the difference between the potential on the internal face and the potential on the external face of the membrane.

We are ready now to introduce the formal notion of triggering conditions that allow the activation of transport proteins.

Definition 13. For each $A \in V$, the *triggering condition* of A, denoted by $tr(A)$, is a 3-tuple $\tau = (n_1, n_2, n_3) \in (\mathbb{N}^{\alpha} \cup \{\dagger\}) \times (\mathbb{N} \cup \{\dagger\}) \times \mathbb{N}$.

The intuition behind this definition is as follows:

- n_1 is called the *concentration gradient triggering condition*: it represents the threshold values of the concentration gradients $\Delta Conc_{f(A)}$ below (or above) which protein A is active. If $n_1 = \dagger$, then the concentration gradient does not influence the activation of A;
- n_2 is called the *potential gradient triggering condition*: it represents the threshold value of the potential gradient $\Delta Volt$ below (or above) which protein A is active. If $n_2 = \dagger$, then the potential gradient does not influence the activation of A;
- n_3 is called the *energy consumption triggering condition*: it represents the number of energy units (which correspond to ATP molecules in the cell) that are needed for protein A to be active. If $n_3 = 0$, then the protein does not use the energy derived from ATP hydrolysis.

In some cases, the numeric threshold values of concentration or voltage gradients could be replaced by intervals, which determine the range within which the protein is, or isn't, active.

The maximum transport rate of the transport proteins population, which is achieved when each protein is active and functioning at its maximal rate, can be then derived according to (1) the number of transport proteins residing in the membrane at any evolution time, and (2) the concentration or voltage gradients across the membrane, or the number of available energy units.

Example 3. The *sodium-calcium exchanger* E in excitable cells can work either in a direct or in a reverse form, according to the current conditions. In the direct form, its functionality is $f_d(E) = \{(Ca^{2+}, \uparrow, 1), (Na^{+}, \downarrow, 3)\}$ and its triggering condition depends on the intracellular regulatory calcium concentration, which has to be higher than 0.1 μM. In the reverse form, its functionality is

$f_r(E) = \{(Ca^{2+}, \downarrow, 1), (Na^{+}, \uparrow, 3)\}$, and its triggering condition depends on the intracellular sodium concentration, which has to be higher than 100mM, and on the membrane potential, which has to be around -40mV ([32,14]).

Remark 3. In some physiological conditions of living cells, it might be more realistic to consider *non-homogeneous* distribution of ions across and along the membrane. In this case, the previous definitions of membrane gradients have to be modified, in order to describe the *local* values of these parameters. Accordingly, the positions of each transport protein over the membrane surface and the knowledge of its local surrounding conditions, determine the activation of the protein at each evolution time. However, an important role is also played by the effective distance between the membrane and the ions on its internal and external sides. A possible formalization of these situations, deserving further investigation, was proposed in [2] by means of "virtual membranes"; also, fuzzy control methods for the functioning of transport proteins were sketched in [1].

6.2 Membrane Microdomains and Protein Clustering

Microdomains are small areas in cellular membranes where either the lipid and protein composition, or the structure and curvature, are different with respect to the rest of the membrane. At microdomains, complex phase behaviors of the membrane can be seen, such as transient separations between the fluid (liquid-crystalline) normal phase of the membrane, and a liquid-ordered phase. This is the case for *lipid rafts* ([3,10]), small and dynamic microdomains where sphingolipids and cholesterol concentrate. Due to their molecular structure, these lipids transiently get tightly packed and ordered by attractive forces (not normally acting on other types of lipids). Since the bilayer is thicker in the rafts, some transmembrane proteins (having long enough spanning segments) can better accommodate there. In this way, lipid rafts can help to organize these proteins by clustering them together, thus allowing functions such as signaling transduction, secretory and endocytic sorting and trafficking.

Besides the presence of rafts, *protein clustering* is an important mechanism occurring in cells, by which several synergic phenomena can take place. For instance, the "calcium-induced-calcium-release" process in muscle cells happens because of cooperation between clusters of RyR channels (at the sarcoplasmic reticulum membrane) and clusters of dihydropyridine receptors (at the tranverse tubules in the plasma membrane) ([33]). Also the bacterial chemotactic response to attractants or repellents is mediated by clustered proteins, involved in a complex signal transduction pathway between protein receptors and the flagellar motors ([5,20]).

Membrane microdomains of a different type are represented by *caveolae*, which are small and specialized invaginations of the plasma membrane, implicated in endocytosis, signal transduction and lipid trafficking ([10]). Caveolae are dynamic structures whose formation seems to be ruled by caveolin, a protein that is also assumed to play a role in vesicle formation. Experimental evidence suggest that, similarly to lipid rafts, a variety of cell-surface signaling pathways are concentrated in caveolae.

The presence of membrane microdomains could be described in the framework of the parametric spherical membranes by defining a set $\mathcal{D}$ of (free or occupied) adjacent intersection points, spanning a portion of some parallels and some meridians. In the local area given by $\mathcal{D}$, the operations defining the movement of proteins could then be applied in a different way: for instance, reduced application rates can be used to characterize lipid rafts, or even no application at all can be considered to represent the trapping of integral proteins in specific positions. On the other hand, the formation of a microdomain, such as rafts or caveolae, should result as an emergent property of the movements and interactions of membrane proteins, possibly by considering the local lipidic composition of the membrane as well.

The occurrence of proteins of the same – or chemically affine – type in a set $\mathcal{D}$, due to either random or preferential movements and interactions of the proteins, can describe the presence or the formation of protein clusters. In this case, the effective functioning of the clustered proteins can be different with respect to their functioning in isolated positions, due to their synergic and feedback interactions.

6.3 Membrane Curvature

The cell shape is the result of many physical forces operating on the membrane. In some cases, the shape is stabilized and permanent, for example in microvilli or in the dendritic tree; in other cases, the conformation of cellular membranes can change considerably, for instance during processes such as movement, division, vesicular trafficking, etc. Membrane curvature is generated by a complex interplay between lipids, membrane proteins, the tensional forces that are applied to the membrane surface, and cell's sensors that feed back to the production of specific curvature-related molecules.

There are several curvature generating mechanisms, possibly working in synergy at the cell membranes. Besides the lipid composition and the role played by the cytoskeleton elements, many remodeling actions are induced by membrane proteins. This is the case in the formation of highly curved vesicles, where the membrane curvature is the effect of polymerized peripheral proteins, called coat proteins (e.g., clathrin), sometimes linked to membranes through other adaptor proteins. Coat proteins form a sort of exoskeleton around the vesicle, and they induce and regulate the membrane traffic for intracellular transport of molecules. Experimental studies of clathrin-coated invaginations revealed that actually several different proteins work together to promote membrane bending and vesicle formation ([25]).

Other scaffolding mechanisms ([38]) due to membrane proteins are the results of banana-shaped protein domains (e.g., BAR domain) found in a wide variety of proteins (see, e.g., [31]). These domains bind to membrane lipids through their intrinsic concave surface, thus remodeling and stabilizing the membrane curvature. Another way to increase the positive membrane curvature is by the insertion of amphipathic protein helices into the bilayer.

The selective binding of proteins to membrane, depending on its curvature, and the partitioning of lipids into curvature-changing regions, also gives the possibility of creating local microenvironments on the membrane (see also Section 6.2). For instance, specific membrane curvature values could help either in the segregation of transmembrane proteins to incorporate them in vesicles or in membrane tubules, or even in the preferential localization of ion channels in membrane protrusions.

The formal description of curvature-generating mechanisms is not a trivial task. Indeed, many physical and biological aspects of the interactions between proteins and the membrane should be considered. For instance, the structure and elasticity properties of the cellular membrane, and the electrostatic interactions and chemical bonds among proteins, play an important role in the formation of local (sometimes transient) areas with differential curvature. Hence, these properties should be incorporated into the parametric 2-dimensional membrane framework, in order to characterize the changes in the surface curvature as a consequence of membrane and proteins interactions. A possible way to do this is to associate to specific neighbor proteins a coefficient that describes the "strength" of their bond; then, assume that, when this force is strong enough (is above some threshold), a positive or negative curvature modification (invagination or extroversion) takes place at the local area where these proteins reside upon the membrane surface.

An example of highly modified curvature occurring in small local areas of the surface membrane is given by the creation of budding vesicles, involved in endocytosis or exocytosis processes. When this is due to coat proteins (hence to the clustering and binding of many peripheral proteins of the same type), then this can be formalized with the parametric membrane surface. For instance, one might define the additional operation of *encapsulation*: it should involve the occurrence of many membrane proteins of the same type residing in adjacent intersection points (e.g., occupying a predefined set $\mathcal{D}$). Whenever such a local condition is satisfied, then the portion of the surface corresponding to $\mathcal{D}$ is "removed" from the membrane, thus simulating the budding of a vesicle. When considering also communication features, and the presence of internal and external multisets, then the vesicles can be used to transport objects from a membrane to an adjacent (internal or external) one.

Finally, we remark that in the study of curvature generating mechanisms, the formation of cell shape due to cytoskeletal elements can be more challenging to formalize – most probably this would require an additional extension of the framework presented in this paper.

6.4 New Conceptualizations of Membrane Architecture

The seminal "fluid mosaic model" ([35]) was based on the principle of Brownian motion, which explains molecular diffusion as the macroscopic effect of thermal agitation processes, by which molecules are always moving around and colliding with each other. Recent experimental evidence indicated that the lateral

movements of membrane components is not so free, but constrained by various mechanisms and phenomena, such as the presence of membrane microdomains or interactions with cytoskeleton elements. Indeed, several aspects of membrane dynamics cannot be explained by the Singer-Nicolson model: e.g., (1) the considerably smaller (by factors of 5 to 50) diffusion coefficients of plasma membrane with respect to those in artificial membranes, or (2) the reduced diffusion or immobilization (after the formation) of oligomers or other big molecular complexes.

Therefore, new models for membrane organization have been recently proposed in [22,24] suggesting that the membrane is "more mosaic than fluid" ([17]) and it can be seen as a compartmentalized fluid where proteins have differential interactions in a crowded environment, the membrane thickness is variable and diffusion is not due to a pure Brownian motion. Light microscopy methodologies are currently used for studying such dynamic processes in living cells. In particular, high-speed single-molecule tracking methods (with nanometer-level precision and smallest time resolution) helped in observing actual movements of lipids and proteins upon the plasma membranes of many types of cells ([22]). The entire plasma membrane – except for clathrin-coated pits, microvilli, cell-cell and cell-substrate junctions – is partitioned into many submicron-sized compartments where molecules undergo short-term confined diffusion (within a compartment), and long-term hop diffusion between adjacent compartments (where molecules become again temporarily trapped)[3]. Moreover, the fact the it takes time to hop from a compartment to an adjacent one, can explain the high disparity between diffusion coefficients in the plasma membrane and diffusion coefficients in reconstituted membranes.

Rafts and cytoskeleton are the two main compartmentalizing forces at work in the plasma membrane. In [22], the *membrane-skeleton fence model* and the *anchored-transmembrane protein pickets model* are introduced to discuss the role of cytoskeleton in membrane compartmentalization. According to these models, the cytoskeleton actin elements form the "fences", and the transmembrane proteins anchored to the cytoskeleton form the "pickets". The fluctuating lattice formed by transmembrane proteins and actin-based skeleton elements creates the barriers which restrict the lateral diffusion of membrane molecules. The hopping movement between compartments happens when, due to thermal fluctuations, a transient gap is formed between the membrane and the cytoskeleton thus allowing the passage of the cytoplasmic domain of moving transmembrane proteins.

The two models can also give reason of the reduced diffusion rate, or immobilization, of membrane molecules upon oligomerization or complex formation. Indeed, monomers can hop across the fence barriers with relative ease, but the complexes as a whole have to hop all at once and hence they are characterized by a much slower rate of hopping between the compartments. It is also suggested that this *oligomerization-induced trapping* might be important in signaling transduction, by temporary confining cytoplasmic signals to the place

[3] The fluid mosaic model agrees with this new paradigms when limited to the events occurring in membrane areas of 10 nm × 10 nm dimension.

where the extracellular signal was received. We refer to reviews [22] and [24] for further details concerning new conceptualizations of membrane architecture.

The fence and pickets models can be formalized within the framework of the parametric surface membrane, even without considering a formal description of the cytoskeleton. Indeed, its role in the formation of fences can be simulated by singling out the "borders" of the compartmentalized lattice over the membrane surface (by identifying them with sub-portions of parallels and meridians). The proteins residing upon the intersection points characterizing the lattice constitute the pickets.

Lattice and protein pickets positions could also be considered as varying in time, but on a slower time scale with respect to the movement of proteins upon the membrane surface due to the application of the operations defined in Section 4. Moreover, the movement of proteins inside a fenced compartment should be characterized by a rate different from the one corresponding to the hop movements between adjacent compartments.

Finally, to have a formal description of the oligomers formation, the assumption of self-avoidance sharing at intersection points has to be dropped. In this case, the monomers are to be treated in a different way, allowing their co-presence at an intersection point. As before, the rate of movement of an oligomer can then be modified to take into account the fact that it is more difficult for an oligomer to change its position on the surface.

6.5 Computational Aspects of the Parametric Membrane Surface

The structure and functioning of membranes is central both for membrane computing (see, e.g., [28] and [30]) and brane calculi (see, e.g., [11]). In membrane computing (which was developed much earlier) the objects (molecules) are processed in regions which are delimited by membranes, while in brane calculi the objects are processed on membranes.

A number of recent papers, beginning with [12], consider a merger of the two approaches. In [12], P systems based on brane operations were considered: objects are now placed on the membrane, as a natural correspondence to membrane proteins. Then, in [15] the projective version of brane calculi was proposed, where proteins are placed and active, or visible, only *on one side* of the membrane. A similar perspective, taking inspiration from the distinction between integral and peripheral proteins, was considered also in [8]. Several other papers associating protein objects to the membranes recently appeared; an up-to-date bibliography of these topics can be found at the P Systems Web Page: `http://psystems.disco.unimib.it`.

In (mem)brane systems, usually *multisets* of objects representing proteins are associated with the membranes. Moreover, as indicated several times in the previous sections, in most of cellular processes occurring on membranes the position of proteins is important. To account for this, a (*circular*) *string* of protein objects – rather than a multiset – can be associated with each membrane, providing in this way a formal structure where the position of each object is known and fixed with respect to all other objects appearing in the string. Then, the parametric

membrane surface can be seen as a 2-dimensional extension of this formal representation where, given any object residing on a parallel (the circular string), one knows not only its left and right neighbors, but also its upper and lower (as well as diagonal) neighbors.

The use of circular words and operations for protein movement, either on 1-dimensional or on 2-dimensional membranes, provides an alternative approach to (mem)brane systems for the formal expression of symbolic membrane proteins, and thus for the analysis of the computational properties of the corresponding systems.

Another suggestion for the analysis of computational topics comes from the definition of functionality of transport proteins. Usually, in the area of membrane computing, the communication of objects across membrane occurs in a non-selective way (any object type can be moved from one region to an adjacent one, unless some restrictions are imposed). Also one allows the passage of an unbounded number of objects at each step (due to the maximal parallelism at the level of rule application). But, as we have seen in Section 5, each transport protein allows only a prescribed number of objects to cross the membrane, and moreover this movement happens in a specific direction and for specific types of objects. Hence, new communication rules could be defined in membrane systems, taking care of the number and the types of the transport proteins that are present on each membrane (and this number can be modified by allowing also insertion or deletion operations). An immediate consequence would be the "bounded and selective" communication of objects.

This approach can be considered both for the 1-dimensional concept of membrane and for the 2-dimensional surface. Formal language generative power or decidability properties, as well as other computational aspects, can then be analyzed in membrane systems with these additional features.

7 Final Remarks

Several aspects of cellular membranes as well as possible applications to biological systems or computational models motivated this paper.

First of all, we have pointed out why the notion of membrane in P systems cannot account for the concept of spatial neighborhood upon a real membrane. At the same time, we have explained why this concept is important when considering "objects" placed within/upon the membrane, such as MP populations. Therefore, we have proposed to extend the notion of membrane to a 2-dimensional spherical surface, whose parametrization defines a scaffolding grid for protein placements. Then, operations acting on proteins can allow either their movement upon the spherical membrane, or their insertion or removal from the membrane.

As a first step, we have proposed to use a spherical surface, parameterized by means of parallels and meridians. However, it is known that living cells can have very different shapes, thus other geometrical surfaces can be utilized for a better description of processes occurring upon the plasma membrane of real cells. Moreover, also different parameterizations, giving rise to specific tessellations of

the surface, can be used as well. An interesting application would be to derive, starting from the spherical surface, other distinct shapes emerging from local and global dynamics (of proteins) on the membrane surface.

The formalization of the spherical membrane has been given for *one* membrane only, but it could also be implemented as a part of a "modified" membrane system. In this case, the (extended) notion of *membrane structure* would then consists of a collection of spherical parametric membranes, contained inside a unique external spherical membrane – the plasma membrane. In an extended membrane structure, the concept of mutual position between spherical membranes, together with their orientation in the 3-dimensional space, would have to be defined as well.

Possible applications of the spherical membrane and MP populations have been discussed for both biological and computational oriented analysis. In particular, the feasibility of this framework in biology also highlights the coexistence of both a systemic approach and a (supra)molecular description of basic system components (e.g., in the investigation of the ion flows occurring at the whole plasma membrane – the systemic analysis, together with the characterization of functioning and activation of single transport proteins – the basic elements description).

Throughout the paper we have mostly referred to transmembrane proteins, and focused the attention on the description of transport proteins. As a matter of fact, the parametric membrane could be used to represent also the placement (or the induced movements) of peripheral membrane proteins populations. This can be easily done, without an essential change of the formal structure, just by removing the self-avoidance sharing requirement. This extension would allow the formalization of complex formation between integral and peripheral proteins, possibly also considering the electrostatic interactions or other physical forces depending on local membrane composition and density, as already discussed in Section 6.

Acknowledgment. This work has been supported by the European Research Training Network "Segravis". We are indebted to Prof. H. Spaink for useful discussions at the beginning phase of this research paper.

References

1. S. Aguzzoli, I.I. Ardelean, D. Besozzi, B. Gerla, C. Manara, P systems under uncertainty: the case of transmembrane proteins, *Proceedings of Brainstorming Workshop on Uncertainty in Membrane Computing*, Palma de Mallorca, 8-10 November 2004, 107–117.
2. S. Aguzzoli, D. Besozzi, B. Gerla, C. Manara, P systems with vague boundaries: the t-norm approach, *Proceedings of Brainstorming Workshop on Uncertainty in Membrane Computing*, Palma de Mallorca, 8-10 November 2004, 97–105.
3. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter, *Molecular Biology of the Cell.*, 4th edition, Garland Science, New York, 2002.

4. I.I. Ardelean, D. Besozzi, On modeling ion fluxes across biological membranes with P systems, *Proceedings of the Third Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan eds.), RGNC Report 01/2005, Sevilla, January 31 - February 4, 2005, 35–42.
5. I.I. Ardelean, D. Besozzi, Some notes on the interplay between P systems and chemotaxis in Bacteria, *Fourth Brainstorming Week on Membrane Computing*, Sevilla, January 30 - February 3, 2006, Volume I (M.A. Gutiérrez-Naranjo, G. Păun, A. Riscos-Núñez, F.J. Romero-Campero eds.), RGNC REPORT 02/2006, Fénix Editora, Sevilla (2006), 41–48.
6. I.I. Ardelean, D. Besozzi, M.H. Garzon, G. Mauri, S. Roy, P system models for mechanosensitive channels. In: G. Ciobanu, G. Păun, M.J. Pérez-Jiménez eds., *Applications of Membrane Computing*, Springer–Verlag, Berlin, 2005.
7. P.W. Atkins, L.L. Jones, *Chemistry: molecules, matter, and change.* Third Edition, W.H. Freeman and Co., New York, 1997.
8. D. Besozzi, N. Busi, G. Franco, R. Freund, G. Păun, Two universality results for (mem)brane systems, *Fourth Brainstorming Week on Membrane Computing*, Sevilla, January 30 - February 3, 2006, Volume I (M.A. Gutiérrez-Naranjo, G. Păun, A. Riscos-Núñez, F.J. Romero-Campero eds.), RGNC REPORT 02/2006, Fénix Editora, Sevilla (2006), 49–62.
9. D. Besozzi, G. Ciobanu, A P system description of the sodium-potassium pump, *Membrane Computing, 5th International Workshop - WMC 2004* (G. Mauri, G. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa eds.), LNCS 3365, Springer-Verlag, Berlin, 2005, 210–223.
10. D.A. Brown, E. London, Functions of lipid rafts in biological membranes, *Annu. Rev. Cell Dev. Biol.*, 14, 1998, 111–136.
11. L. Cardelli, Brane calculi. Interactions of biological membranes, *Computational Methods in Systems Biology. International Conference CMSB 2004*, Paris, France, May 2004, Revised Selected Papers (V. Danos, V. Schachter, eds.), LNCS 3082, Springer-Verlag, Berlin, 2005, 257–280.
12. L. Cardelli, G. Păun, An universality result for (mem)brane calculus based on mate/drip operations, *International Journal of Foundations of Computer Science*, 17, 1, 2006, 49–68.
13. W. Cho, R.V. Stahelin, Membrane-protein interactions in cell signaling and membrane trafficking, *Annu. Rev. Biophys. Biomol. Struct.*, 34, 2005, 119–151.
14. F. Cossu, *Modelli discreti per il trasporto del calcio attraverso la membrana plasmatica*, Graduation Thesis, University of Milano, Italy, 2005.
15. V. Danos, S. Pradalier, Projective brane calculus, *Computational Methods in Systems Biology: International Conference CMSB 2004*, Paris, France, May 26-28, 2004, Revised Selected Papers, (V. Danos, V. Schachter, eds.), LNCS 3082, Springer-Verlag, Berlin, 2005, 134–148.
16. M. Dean, *The Human ATP-Binding Cassette (ABC) Transporter Superfamily*, National Library of Medicine (US), NCBI, 2002 (http://www.ncbi.nlm.nih.gov/).
17. D.M. Engelman, Membranes are more mosaic than fluid, *Nature*, 438, 2005, 578–580.
18. M. Fill, J.A. Copello, Ryanodine receptors calcium release channels, *Physiol. Rev.*, 82, 2002, 893–922.
19. E. Gouaux, R. MacKinnon, Principles of selective ion transport in channels and pumps, *Science*, 310, 2005, 1461–1465.
20. M.S. Jurica, B.L. Stoddard, Mind your B's and R's: bacterial chemotaxis, signal transduction and protein recognition, *Current Biology*, 6, 1998, 809–813.

21. C. Kosniowski, *A First Course in Algebraic Topology*, Cambridge University Press, 1980.
22. A. Kusumi, C. Nakada, K. Ritchie, K. Murase, K. Suzuki, H. Murakoshi, R.S. Kasai, J. Kondo, T. Fujiwara, Paradigm shift of the plasma membrane concept from the two-dimensional continuum fluid to the partitioned fluid: high-speed single-molecule tracking of membrane molecules, *Annu. Rev. Biophys. Biomol. Struct.*, 34, 2005, 351–378.
23. H. Lodish, A. Berk, S.L. Zipursky, P. Matsudaira, D. Baltimore, J.E. Darnell, *Molecular Cell Biology*. 4th Ed., W.H. Freeman and Co., New York, 2000.
24. D. Marguet, P.F. Lenne, H. Rigneault, H.T. He, Dynamics in the plasma membrane: how to combine fluidity and order, *The EMBO Journal*, 25, 2006, 3446–3457.
25. H.T. McMahon, J.L. Gallop, Membrane curvature and mechanisms of dynamic cell membrane remodelling, *Nature*, 438, 2005, 590–596.
26. N. Palmieri, *Un approccio stocastico alla modellazione del canale RyR*, Graduation Thesis, University of Milano, Italy, 2006.
27. M.J. Pérez-Jiménez, F.J. Romero-Campero, Modelling EGFR signalling cascade using continuous membrane systems. *Proceedings of CMSB2005* (G. Plotkin, ed.), Edinburgh, 3-5 April 2005, 118–129.
28. G. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143.
29. G. Păun, Computing with membranes – A variant: P systems with polarized membranes, *International Journal of Foundations of Computer Science*, 11, 1, 2000, 167–182.
30. G. Păun, *Membrane Computing. An introduction*, Springer–Verlag, Berlin, 2002.
31. G.A. Petsko, D. Ringe, *Protein Structure and Function*, New Science Press Ltd., 2004.
32. K.D. Philipson, D.A. Nicoll, Sodium-calcium exchange: a molecular perspective, *Annual Review of Physiology*, 62, 2000, 111–133.
33. I.I. Serysheva, Structural insights into excitation-contraction coupling by electron cryomicroscopy, *Biochemistry* (Moscow), 69, 11, 2004, 1226–1232.
34. S.J. Singer, Some early history of membrane molecular biology, *Annual Review of Physiology*, 66, 2004, 1–27.
35. S.J. Singer, G.L. Nicolson, The fluid mosaic model of the structure of cell membranes, *Science*, 175, 1972, 720–731.
36. J.M. Ward, Patch-clamping and other molecular approaches for the study of plasma membrane transporters demystified, *Plant Physiology*, 114 (1997), 1151–1159.
37. W. Wickner, R. Schekman, Protein translocation across biological membranes, *Science*, 310, 2005, 1452–1456.
38. J. Zimmerberg, M.M. Kozlov, How proteins produce cellular membrane curvature, *Nature Reviews Molecular Cell Biology*, 7, 2006, 9–19.

Quorum Sensing: A Cell-Cell Signalling Mechanism Used to Coordinate Behavioral Changes in Bacterial Populations

Miguel Cámara

Institute of Infection, Immunity and Inflammation
Centre for Biomolecular Sciences
University of Nottingham, Nottingham NG7 2RD, UK
miguel.camara@nottingham.ac.uk

1 The Quorum Sensing Concept

One of the most important mechanisms for bacterial cell-to-cell communication and behavior coordination under changing environments is often referred to as "quorum sensing" (QS). QS relies on the activation of a sensor kinase or response regulator protein by, in many cases, a diffusible, low molecular weight signal molecule (a "pheromone" or "autoinducer") (Cámara et al., 2002). Consequently, in QS, the concentration of the signal molecule reflects the number of bacterial cells in a particular niche and perception of a threshold concentration of that signal molecule indicates that the population is "quorated", i.e. ready to make a behavioral decision. Bacteria cell-to-cell communication is perhaps the most important tool in the battle for survival; they employ communication to trigger transcriptional regulation resulting in sexual exchange and niche protection in some cases, to battle host' defences and coordinate population migration. Ultimately, bacteria cell-to-cell communication is used to effect phenotypic change. The importance of coordinated gene-expression (and hence phenotypic change) in bacteria can be understood if one realizes that only by pooling together the activity of a quorum of cells can a bacterium be successful. It is increasingly apparent that, in nature, bacteria function less as individuals and more as coherent groups that are able to inhabit multiple ecological niches (Lazdunski et al., 2004). Within quorum sensing process several key elements must be considered: (i) the gene(s) involved in signal synthesis, (ii) the gene(s) involved in signal transduction, and (iii) the QS signal molecule(s).

In Gram-negative bacteria, some of the most studied signal molecules are the *N*-acylhomoserine lactones (AHLs) (Fig. 1A). During the growth of a bacterial population, signal molecules either diffuse or are exported out of the cell into the surrounding environment; their concentration increases and they then act on neighboring bacterial cells. Achievement of a critical threshold concentration results in: (i) activation of a sensor/response regulator, responsible for signal transduction (T), which in turn triggers the expression of multiple genes and (ii) activation positive autoinductive feedback loop to amplify QS signal molecule generation (Figure 1B).

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 42–48, 2006.

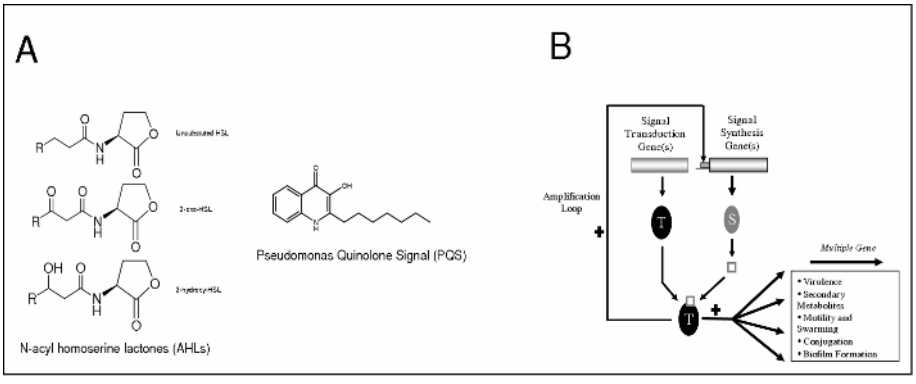


Fig. 1. A: Chemical structures of *N*-acylhomoserine lactones and PQS. B: Generic quorum sensing signal generation and transduction circuit.

Hence the term "autoinducer" is sometimes used to describe the QS signal molecule. Important in governing the size of the "quorum" is 'compartment sensing' (Winzer et al., 2002). The concentration of a given QS signal molecule may be a reflection of bacterial cell number, or at least the minimal number of cells (quorum) in a particular physiological state. To achieve the accumulation of a QS signal there is a need for a diffusion barrier, which ensures that more molecules are produced than lost from a given microhabitat. This 'compartment sensing' enables the QS signal molecule to be both a measure of the degree of compartmentalization and the means to distribute this information through the entire population. Likewise, the diffusion of QS signal molecules between detached subpopulations may convey information about their numbers, physiological state and the specific environmental conditions encountered.

2 Quorum Sensing in *P. aeruginosa*: A Complex Regulatory Network Resulting in Fine Signal Tuning

Pseudomonas aeruginosa is a very versatile organism that can adapt to many different environments and can cause diseases in plants, animals, and humans (Rahme et al., 1995). It possesses a large 6.3MB genome encoding 5,570 predicted genes including 521 putative regulatory genes suggesting the existence of a highly complex gene regulation which enables it to adapt quickly to environmental changes (Stover et al., 2000). This organism produces a broad range of exoproducts, which are regulated in a population density-dependent manner via cell-to-cell communication or "quorum sensing" (Cámara et al., 2002). Two intertwined QS systems (the *las* and the *rhl* systems) have been shown to be involved in virulence, biofilm development, and many other processes in *P. aeruginosa* (Gambello and Iglewski, 1991; Latifi et al., 1995; Ochsner and Reiser, 1995; Passador et al., 1993). These QS systems produce and respond to specific AHL signal molecules (Pearson et al., 1994; Winson et al., 1995). In addition,

each system modulates a regulon comprising an overlapping set of genes. However, the *las* and the *rhl* systems are not independent of each other, but form a regulatory hierarchy where LasR-C12-HSL activates the transcription of *rhlR* (Latifi et al., 1996; Pesci et al., 1997) (Figure 2). Transcriptome analysis of *P. aeruginosa* has revealed that *N*-acylhomoserine lactone (AHL)-dependent QS regulates up to 10% of the genes in the genome of this organism (Schuster et al., 2003; Whiteley et al., 1999).

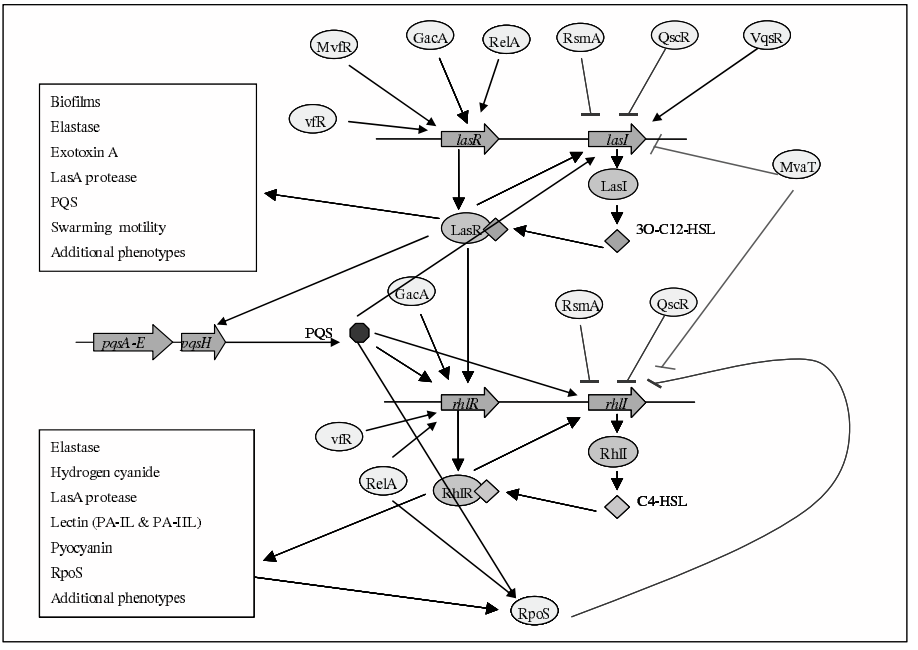


Fig. 2. Interactions between the different regulators of QS in *P. aeruginosa* (→ indicates positive regulation and ⊣ negative regulation)

In addition to AHLs, *P. aeruginosa* releases a 4-quinolone signal molecule into the extra-cellular milieu, the synthesis and bioactivity of which has been reported to be mediated via the *las* and *rhl* systems respectively. This molecule has been chemically identified as 2-heptyl-3-hydroxy-4(1H)-quinolone and termed the Pseudomonas Quinolone Signal (PQS) (Fig. 1) (Pesci et al., 1999). *LasR* has been shown to regulate PQS production and the provision of exogenous PQS induces expression of *lasB* (coding for elastase), *rhlI* and *rhlR* (McKnight et al., 2000; Pesci et al., 1999) suggesting that PQS activity constitutes a regulatory link between the *las* and *rhl* QS systems. The QS-dependent production of exoproducts in *P. aeruginosa* is tightly regulated with respect to growth phase and growth environment. In contrast to the AHL-dependent induction of bioluminescence in *Vibrio fischeri* (Eberhard et al., 1981) and carbapenem antibiotic

production in *Erwinia carotovora* (Williams et al., 1992), the provision of exogenous AHLs does not advance the expression of several QS dependent genes in wild type *P. aeruginosa* PAO1 such as *lecA, lasB* or *rhlR* expression (Diggle et al., 2002; Pearson, 2002). This is due to the contribution of additional regulatory factors in addition to *LasR* and *RhlR*. Figure 2 shows a simplified diagram of how the different regulators have so far been shown to interact with the QS regulatory cascade at both the transcriptional and posttranscriptional level. This shows an example of the intricate control of QS-mediated responses by a network of regulators which results in a fine tuning of adaptive responses to environmental changes.

3 Potential Approaches for Systems Biology-Based Study of Regulatory Networks in *P. aeruginosa*

To gain a better understanding on how regulatory networks in *P. aeruginosa*, or any other organisms using quorum sensing-mediated signalling mechanisms, work, there are a number of key questions that need to be address: (i) what are the key parameters governing the relationship between QS master switches? (ii) how do cellular regulatory networks fine tune into the QS regulatory cascade? (iii) for genes directly regulated by more than one regulatory system what are the rules that determine the establishment of the hierarchical control? To answer these questions two possible systematic approaches could be used. On one hand, the regulatory networks could be analyzed using a "zoom in" model, starting by studying phenotypic changes and eventually identifying their tuning with key cellular regulators (Figure 3). In this particular model a bacterial population could be subjected to certain inputs determined by changes in environmental conditions. This would result in phenotypic changes which could be measured. If the biology of the organism is reasonably known, these changes could be linked to transcriptional alterations of previously characterized target genes. Subsequently, using a number of genetic tools such as those described by Diggle et al for *P. aeruginosa* (Diggle et al., 2002), novel regulators for those target genes could be identified. The validation of this model could be done by testing whether changes in environmental conditions, identical to those used before, would result in the same alteration on the activity of the transcriptional regulators identified, their target genes and the corresponding phenotypes.

Alternatively, a "zoom out" model going from the understanding of the relationship between core regulators to the way they influence phenotypic changes in a bacterium. This approach, shown in Figure 4 would start by investigating the relationship between known core regulators such us the *rhl*, *las* or PQS systems in *P. aeruginosa*, and their supraregulators. The next step up would require moving out one layer and investigating the effect alterations in these regulators would have on the expression of their target genes resulting in specific phenotypic changes which could be measured. Validation of this model could be done by verifying that specific alteration to the activity of the regulators results in

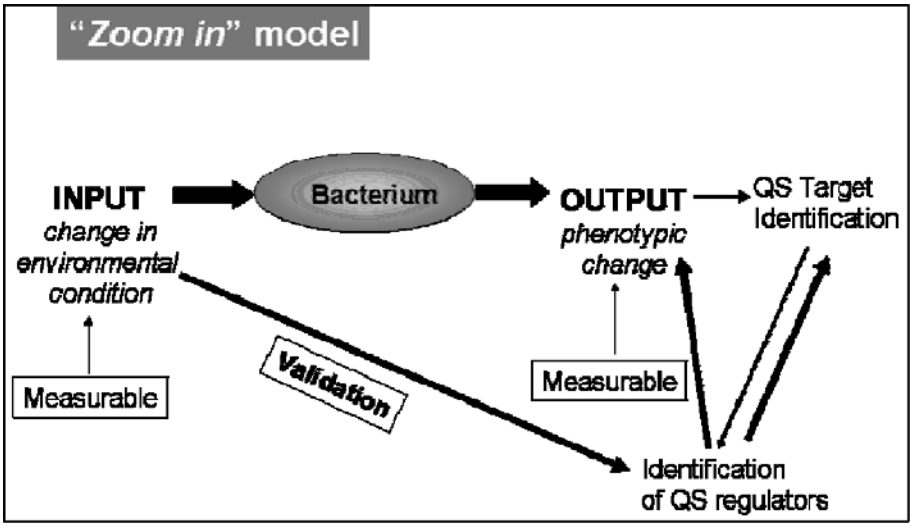


Fig. 3. Systems-biology based "Zoom in" approach to study regulatory networks in *P. aeruginosa*

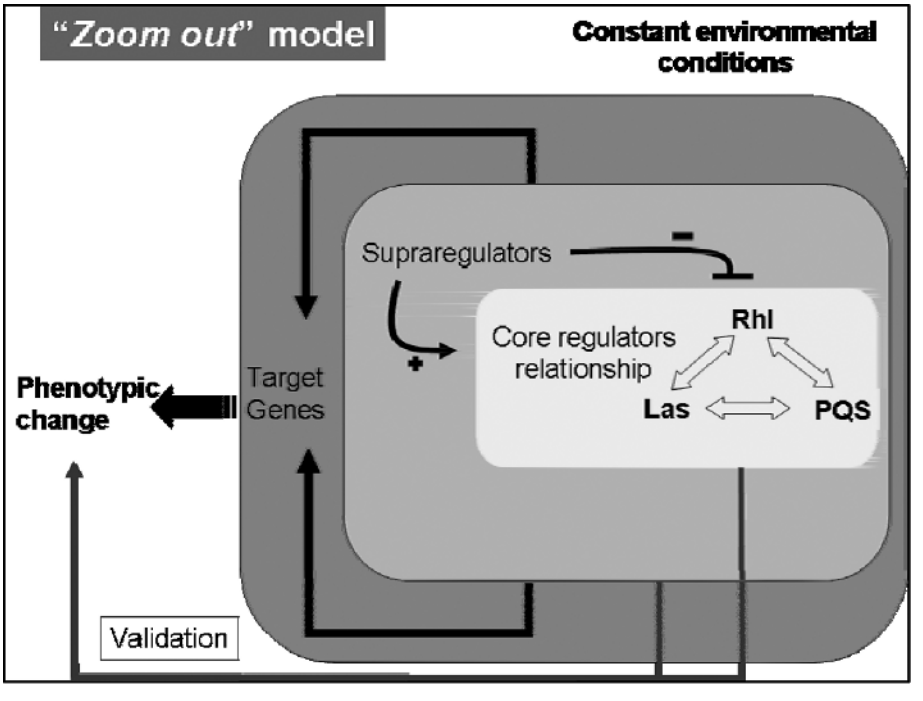


Fig. 4. Systems-biology based "Zoom out" approach to study regulatory networks in *P. aeruginosa*

the predicted phenotypic changes. This type of approach could be simplified by using constant environmental conditions.

Although other systematic models could be used to study the complexity of *P. aeruginosa* QS regulatory networks, these two approaches are perhaps the most straight forward ones.

References

1. Cámara, M., Williams, P., and Hardman, A. (2002) Controlling infection by tuning in and turning down the volume of bacterial small-talk. *Lancet Infectious Diseases* 2: 667–676.
2. Diggle, S.P., Winzer, K., Lazdunski, A., Williams, P., and Cámara, M. (2002) Advancing the quorum in Pseudomonas aeruginosa: MvaT and the regulation of N-acylhomoserine lactone production and virulence gene expression. *Journal of Bacteriology* 184: 2576–2586.
3. Eberhard, A., Burlingame, A.L., Eberhard, C., Kenyon, G.L., Nealson, K.H., and Oppenheimer, N.J. (1981) Structural identification of autoinducer of Photobacterium-Fischeri luciferase. *Biochemistry* 20: 2444–2449.
4. Gambello, M.J., and Iglewski, B.H. (1991) Cloning and characterization of the *Pseudomonas aeruginosa* LasR gene, a transcriptional activator of elastase expression. *Journal of Bacteriology* 173: 3000–3009.
5. Latifi, A., Foglino, M., Tanaka, K., Williams, P., and Lazdunski, A. (1996) A hierarchical quorum sensing cascade in *Pseudomonas aeruginosa* links the transcriptional activators LasR and RhIR (VsmR) to expression of the stationary-phase sigma factor RpoS. *Molecular Microbiology* 21: 1137–1146.
6. Latifi, A., Winson, M.K., Foglino, M., Bycroft, B.W., Stewart, G., Lazdunski, A., and Williams, P. (1995) Multiple homologs of LuxR and Luxl control expression of virulence determinants and secondary metabolites through quorum sensing in *Pseudomonas aeruginosa* PAO1. *Molecular Microbiology* 17: 333–343.
7. Lazdunski, A.M., Ventre, I., and Sturgis, J.N. (2004) Regulatory circuits and communication in gram-negative bacteria. *Nature Reviews Microbiology* 2: 581–592.
8. McKnight, S.L., Iglewski, B.H., and Pesci, E.C. (2000) The Pseudomonas quinolone signal regulates *rhl* quorum sensing in *Pseudomonas aeruginosa. Journal of Bacteriology* 182: 2702–2708.
9. Ochsner, U.A., and Reiser, J. (1995) Autoinducer-mediated regulation of rhamnolipid biosurfactant synthesis in *Pseudomonas aeruginosa. Proceedings of the National Academy of Sciences of the United States of America* 92: 6424–6428.
10. Passador, L., Cook, J.M., Gambello, M.J., Rust, L., and Iglewski, B.H. (1993) Expression of *Pseudomonas aeruginosa* virulence genes requires cell-to-cell communication. *Science* 260: 1127–1130.
11. Pearson, J.P. (2002) Early activation of quorum sensing. *Journal of Bacteriology* 184: 2569–2571.
12. Pearson, J.P., Gray, K.M., Passador, L., Tucker, K.D., Eberhard, A., Iglewski, B.H., and Greenberg, E.P. (1994) Structure of the autoinducer required for expression of *Pseudomonas aeruginosa* virulence genes. *Proceedings of the National Academy of Sciences of the United States of America* 91: 197–201.
13. Pesci, E.C., Milbank, J.B.J., Pearson, J.P., McKnight, S., Kende, A.S., Greenberg, E.P., and Iglewski, B.H. (1999) Quinolone signaling in the cell-to-cell communication system of *Pseudomonas aeruginosa. Proceedings of the National Academy of Sciences of the United States of America* 96: 11229–11234.

14. Pesci, E.C., Pearson, J.P., Seed, P.C., and Iglewski, B.H. (1997) Regulation of *las* and *rhl* quorum sensing in *Pseudomonas aeruginosa. Journal of Bacteriology* 179: 3127–3132.
15. Rahme, L.G., Stevens, E.J., Wolfort, S.F., Shao, J., Tompkins, R.G., and Ausubel, F.M. (1995) Common virulence factors for bacterial pathogenicity in plants and animals. *Science* 268: 1899–1902.
16. Schuster, M., Lostroh, C.P., Ogi, T., and Greenberg, E.P. (2003) Identification, timing, and signal specificity of *Pseudomonas aeruginosa* quorum-controlled genes: a transcriptome analysis. *Journal of Bacteriology* 185: 2066–2079.
17. Stover, C.K., Pham, X.Q., Erwin, A.L., Mizoguchi, S.D., Warrener, P., Hickey, M.J., Brinkman, F.S.L., Hufnagle, W.O., Kowalik, D.J., Lagrou, M., Garber, R.L., Goltry, L., Tolentino, E., Westbrock-Wadman, S., Yuan, Y., Brody, L.L., Coulter, S.N., Folger, K.R., Kas, A., Larbig, K., Lim, R., Smith, K., Spencer, D., Wong, G.K.S., Wu, Z., Paulsen, I.T., Reizer, J., Saier, M.H., Hancock, R.E.W., Lory, S., and Olson, M.V. (2000) Complete genome sequence of *Pseudomonas aeruginosa* PAO1, an opportunistic pathogen. *Nature* 406: 959–964.
18. Whiteley, M., Lee, K.M., and Greenberg, E.P. (1999) Identification of genes controlled by quorum sensing in *Pseudomonas aeruginosa. Proceedings of the National Academy of Sciences of the United States of America* 96: 13904–13909.
19. Williams, P., Bainton, N.J., Swift, S., Chhabra, S.R., Winson, M.K., Stewart, G., Salmond, G.P.C., and Bycroft, B.W. (1992) Small molecule-mediated density-dependent control of gene-expression in prokaryotes - Bioluminescence and the biosynthesis of carbapenem antibiotics. *Fems Microbiology Letters* 100: 161–167.
20. Winson, M.K., Cámara, M., Latifi, A., Foglino, M., Chhabra, S.R., Daykin, M., Bally, M., Chapon, V., Salmond, G.P.C., Bycroft, B.W., Lazdunski, A., Stewart, G., and Williams, P. (1995) Multiple N-acyl-L-homoserine lactone signal molecules regulate production of virulence determinants and secondary metabolites in *Pseudomonas aeruginosa. Proceedings of the National Academy of Sciences of the United States of America* 92: 9427–9431.
21. Winzer, K., Hardie, K.R., and Williams, P. (2002) Bacterial cell-to-cell communication: Sorry, can't talk now - gone to lunch! *Current Opinion in Microbiology* 5: 216–222.

A Modeling Approach Based on P Systems with Bounded Parallelism

Francesco Bernardini[1], Francisco J. Romero-Campero[2], Marian Gheorghe[3], and Mario J. Pérez-Jiménez[2]

[1] Leiden Institute of Advanced Computer Science
University of Leiden
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
bernardi@liacs.nl
[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville, Avda. Reina Mercedes, 41012 Sevilla, Spain
{fran,marper}@cs.us.es
[3] Department of Computer Science, The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
M.Gheorghe@dcs.shef.ac.uk

Abstract. This paper presents a general framework for modelling with membrane systems that is based on a computational paradigm where rules have associated a finite set of attributes and a corresponding function. Attributes and functions are meant to provide those extra features that allow to define different strategies to run a P system. Such a strategy relying on a bounded parallelism is presented using an operational approach and applying it for a case study presenting the basic model of quorum sensing for *Vibrio fischeri* bacteria.

1 Introduction

In 1998, Gheorghe Păun initiated the field of research called membrane computing with a paper firstly available on the web and later published in [19]. Membrane computing aims at defining computational models which abstract from the functioning and structure of the cell. In particular, membrane computing starts from the observation that compartmentalization through membranes is one of the essential features of (eucaryotic) cells. Unlike bacterium, which generally consists of a single intracellular compartment, a(n) (eucaryotic) cell is sub-divided into functionally distinct compartments. Thus, a class of computing devices called membrane systems, or P systems, are defined [19], which have three essential features: a membrane structure consisting of a hierarchical arrangement of several compartments defined as regions delimited by membranes; objects assigned to regions; and rules assigned to the regions of the membrane structure, acting upon the objects inside. In particular, each region is supposed to contain a finite set of rules and a finite multiset (or set) of objects. Rules encode generic processes for producing/consuming objects and for moving objects from one region to the other. Objects are described either as symbols from a

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 49–65, 2006.

given alphabet or as strings over a given alphabet. The application of the rules is performed in a non-deterministic maximally parallel manner: all the applicable rules that should be used to modify existing objects must be applied, and this is done in parallel for all membranes.

Since this model was introduced for the first time in 1998, many variants of membrane systems have been proposed and studied – a comprehensive bibliography of P systems can be found at the P systems web page [29]. The most investigated membrane system topics are related to the computational power of different variants, their capabilities to solve hard problems, like NP-complete problems, decidability, complexity aspects and hierarchies of classes of languages produced by these devices.

Membrane computing represents nowadays a research area of a larger interdisciplinary field called natural computing, that involves scientists studying the emergence of new computational paradigms inspired from the behavior of various natural phenomena. In the same time there is a growing interest in applying mathematical and computational paradigms to model real natural systems. Computational biology is such a field, where mathematical and computational models of biological systems are designed for the analysis and simulation of the behavior of these systems. Biological modeling has involved standard continuous and stochastic mathematical approaches, as well as discrete models. Standard mathematical models with their simulation techniques have proved to be powerful tools for understanding the dynamics of biological systems (e.g., see [25], [26]). Discrete modeling instead advocates the use of different formalisms taken from various areas of computer science (e.g., formal grammars [6], Petri nets [16], X machines [9], [27], process algebra [24], [3], statecharts [15], etc.) to develop computational models of biological systems.

This paper presents a general framework for modeling with membrane systems that is based on a model of P systems where rules have associated a finite set of attributes and a corresponding function. Attributes and functions are meant to provide those extra features which are necessary to close the "gap" between the abstractness of more standard P system models and the "reality" of the phenomenon to be modeled (Section 2). The behavior of such P systems is defined in Section 3 in terms of bounded parallelism, which precisely formalizes the idea of a membrane system as a system where a certain number of components evolve in parallel at the same time by means of a certain number of rules applied inside each one of these components. Then, in Section 4, as a particular instance of the general model, we consider P systems where rules have associated a real constant as an attribute and the corresponding function is used to compute a value of a probability depending on this constant and on certain multisets of objects defining the context where the rule is applied; for this particular variant of P systems, we also define a strategy for the application of the rules which, in each step, selects the next rule to be applied depending on the aforementioned values of probabilities. Finally, in Section 5, as a case study, we present a P system model for the quorum sensing mechanism of bacterial cell-to-cell communication.

2 The Model

Usually a P system is defined as a hierarchical arrangement of a number of membranes identifying a corresponding number of regions inside the system, and with these regions having associated a finite multiset of objects and a finite set of rules. Moreover, one can also consider membrane systems where the underlying structure is defined as an arbitrary graph like in tissue P systems [20], and in population P systems [1]. In this paper, we only focus on membrane systems of the former type where the underlying structure is defined as being a tree of nested membranes.

Rules of many different forms have been considered for membrane systems in order to encode the operation of modifying the objects inside the membranes and the operation of moving objects from one place to the other. In particular, for communicating objects, one can use either the targets $here, in, out$, or symport/antiport rules, or boundary rules [20].

Here, in order to capture the features of most of these rules, we consider rules of the form:

$$u\,[\,v\,] \rightarrow u'\,[\,v'\,] \tag{1}$$

with u, v, u', v' some finite multisets. These rules are generalized boundary rules operating as multiset rewriting rules which simultaneously replace a multiset of objects placed outside the membrane and a multiset of objects placed inside the membrane with two new multisets placed in the same places. In this way, we are able to capture in a concise way the essential features of transformation and communication of objects usually considered in the area of membrane computing. Moreover, from a modeling point of view, rules like (1) allow us to express any sort of interactions occurring at the membrane level, like, for instance, the binding of a signal molecule to a specific receptor which occurs at cell-surface level (e.g., see [21]).

We associate to each rule a finite set of attributes in order to be able to capture specific quantitative aspects of the phenomenon to be modeled. Specifically, these attributes are used by rules as different reaction rates and/or different probabilities which overall affects the strategy of the application of the rules.

Definition 1 (program). *Let V, K be alphabets, let D be a set (possibly infinite), and let A be a finite subset of D. The set D is called the* set of values *and the set A is called the* set of attributes. *Let $\mathcal{F} = \{f_1, f_2, \ldots, f_p\}$ be a finite set of functions such that, for all $1 \leq t \leq n$, $f_t : \mathcal{P}(D) \times V^* \times V^* \times V^* \times V^* \longrightarrow D$ with $\mathcal{P}(D)$ the family of all subsets of D. A* program (over $V, K, D, A, \mathcal{F}$) *is a construct $\langle u\,[\,v\,] \rightarrow u'\,[\,v'\,], \sigma, f_i\rangle_l$ where: $u, v, u', v' \in O^*$, $\sigma \subseteq A$, $f_i \in \mathcal{F}$, for some $1 \leq i \leq p$, and $l \in K$.*

Thus, a program consists of a rule, a finite set of attributes and a function which, given this set of attributes and four multisets, returns a value from a certain chosen set. In particular, the first two multisets are usually considered as being the two multisets placed on the left side of the rule; the two remaining multisets are instead supposed to be the multisets placed respectively inside

and outside a given membrane. That is, each function is expected to compute a particular value depending on the rule and on the contents of the outside and the inside regions that define the "context" where the rule is applied.

Here our main focus is in modeling systems consisting of many different biochemical reactions distributed across different compartments. Therefore, the following interpretation for programs will be predominantly used throughout the paper. Objects represent chemicals and multisets of objects are interpreted as "bags" or "soup" of chemicals. Rules model transformations involving these chemicals. Each rule has associated a finite set of attributes and a corresponding function defining particular properties that affect the behavior of the rule itself; these properties are usually said to define the *reactivity of the rule* (e.g., see [2], [21]). Moreover, since we want to have systems consisting of many different compartments, we assign to each program a label to differentiate the set of programs from one compartment to the other.

Remark 1. Although the aforementioned interpretation will be mainly used in this paper, it is not the unique possible and that is why we introduce the notions of set of values and set of attributes in a more generic fashion. For instance, one may use attributes to model properties inherent to the membranes and may use the corresponding function to update these attributes every time a rule is applied. Alternatively, one may consider hybrid models where the attributes represent continuous variables which are updated through the use of certain functions (e.g., see [9] for hybrid X machines, and [16] for hybrid Petri nets).

Next, we introduce the notion of P specification which makes explicit the basic components necessary to define a particular P system model.

Definition 2 (P specification). *A* P specification *is a construct*

$$\mathcal{S} = (V, K, D, A, \mathcal{F}, P)$$

where:

1. V *is an alphabet; its elements are called* objects*;*
2. K *is an alphabet; its elements are called* labels*;*
3. D *is a* set of values*;*
4. $A \subseteq D$ *is a finite* set of attributes*;*
5. $\mathcal{F} = \{f_1, f_2, \ldots, f_p\}$ *is a finite set of functions as in Definition 1.*
6. P *is a finite set of programs over* $V, K, D, A, \mathcal{F}$ *of the form specified in Definition 1.*

Thus, a P specification provides a "scheme" for the definition of P systems with programs defined over the sets V, K, D, A, and F. In particular, from a modeling point of view, we can say that the alphabet V of objects specifies different "sorts" for the chemicals present inside a certain system, the alphabet K of labels specifies different "types" for the membranes possibly present inside a certain system, and the set D specifies the domain of interpretations for the attributes and the corresponding functions.

A P system is then obtained by augmenting a certain P specification with an initial configuration. In particular, we recall that the structure of a P system is given by a hierarchical arrangement of some $n \geq 1$ membranes labeled in an one-to-one manner with values in $\{1, 2, \ldots, n\}$ [20]. However, since here we use labels from a given alphabet to identify the "type" of a membrane, the value from $\{1, 2, \ldots, n\}$ assigned to a membrane is called the *index of the membrane*; the membrane structure is then said to be *indexed by the values* $1, 2, \ldots, n$.

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

$$\Pi = (\mathcal{S}, \mu, M_1, M_2, \ldots, M_n)$$

where $\mathcal{S} = (V, K, D, A, \mathcal{F}, P)$ *is a P specification with* $V, K, D, A, \mathcal{F}, P$ *as in Definition 2,* μ *a membrane structure containing* n *membranes indexed by the values* $1, 2, \ldots, n$*, and, for all* $1 \leq i \leq n$*,* $M_i = (w_i, l_i)$*, with* $w_i \in V^*$ *the* content *of membrane* i *and* $l_i \in K$ *the* label *of membrane* i*.*

As usual, a P system of degree $n \geq 1$ is defined as consisting of a membrane structure containing n membranes. Each membrane contains a multiset of objects and gets assigned a label from the set K. This latter symbol is particularly useful for retrieving from the given specification the set of programs which can be used inside each membrane in the system. In other words, this label precisely identifies the "type" of the membrane in terms of the rules which can be applied inside.

Most of the P system variants utilize a maximal parallel rewriting manner [20]. This means, in each step, in each membrane, all the objects that can evolve by means of some rules must evolve in parallel, with the only restriction that the same occurrence of the same object cannot be used by more than one rule at a time. That is, in each step, for each membrane, a maximal set of rules to be applied is non-deterministically selected by making sure that no further rules can be applied to the objects left inside the membranes. In this paper we will introduce a mechanism to bound the number of applications of the rules and the number of membranes that will evolve in a step. In this respect, the key issues that need to be addressed in order to define a strategy for the application of the rules in a P system are: a) how to select the next rule to be applied inside a given membrane, b) how many different rules can be applied in parallel at the same time inside the membrane, and c) how many different membranes can evolve in parallel at the same time.

3 Parallelism of Type (k, q)

We formalize here the notion of a transition step in P systems evolving in a $(k, q)-$parallel manner: in each step, at most k membranes evolve in parallel at the same time and, inside each membrane, at most q rules are applied in parallel at the same time. Moreover, in a given step, if k membranes can evolve by means of some rules, then exactly k membranes must evolve in parallel in that step; inside each membrane, if q rules can be applied, then exactly q rules are applied

in parallel inside that membrane. In other words, parallelism of type (k, q) is assumed to be maximal and exhaustive with respect to k and q.

Our formalization makes use of some concepts of the operational semantics for P systems introduced in [5] and it is based on the explicit assignment of the rules contained to the programs to the respective membranes; this is obtained by assigning the index of a membrane to each object possibly present inside the system.

Let $\Pi = (\mathcal{S}, \mu, M_1, M_2, \ldots, M_n)$ with $\mathcal{S} = (V, K, D, A, \mathcal{F}, P)$ and $M_i = (w_i, l_i)$, for all $1 \leq i \leq n$, be a P system as specified in Definition 3. The following extra notions are associated to the P system Π:

- the *indexed alphabet (of Π)* denoted by $\bar{V}$ is the set $\bar{V} = \{ a_i \mid a \in V, 1 \leq i \leq n \}$;
- for all $1 \leq i \leq n$, and for all $u \in V^*$, the *i-version of u* is the multiset denoted by $u_i \in \bar{V}$ such that, for all $a \in V$, for all $1 \leq j \neq i \leq n$, $|u_i|_{a_i} = |u|_a$ and $|u_{a_j}| = 0$;
- the *set of membrane rules (of Π)* is the set of programs denoted by MR and such that: $MR = \{ u_j\, v_i \rightarrow u'_j\, v'_i \mid \langle u\,[\,v\,] \rightarrow u'\,[\,v'\,], \sigma, f \rangle_{l_i} \in P, j = \mathit{upper}(\mu, i) \}$, where $\mathit{upper}(\mu, i)$ is a function returning for a given membrane structure μ, the membrane containing the region i.

Thus, the indexed alphabet of Π is the alphabet of symbols from V with attached indexes of the membranes in the system. The i-version of a multiset u, with $1 \leq i \leq n$ and $u \in V^*$, is the multiset obtained by assigning the index i to all the objects in u. The set MR explicitly identifies, for each membrane, the set of rules which can be used inside that membrane by replacing the multisets in the rules with the corresponding indexed versions. In particular, for all $1 \leq i \leq n$, the set of rules which can be used inside membrane i are the rules contained in programs labeled by l_i.

Moreover, for all $1 \leq i \leq n$, a *multiset of rules for membrane i*, is a collection of membrane rules $r_1, r_2, \ldots, r_{k_i}$, with $k_i \geq 0$ and with these rules not necessarily distinct, such that, for all $1 \leq h \leq k_i$, r_h is a membrane rule in MR of the form $u_j\, v_i \rightarrow u'_j\, v'_i$ with $v_i \neq \lambda$. The *size* of R_i, denoted by $|R_i|$, is the number of rules in R_i.

A *multiset of rules in Π* is a collection of membranes rules of the form R_1, $R_2, \ldots, R_n$ such that, for all $1 \leq i \leq n$, R_i is a multiset of rules for membrane i.

Then, a configuration of a P system is defined as being a multiset over the indexed alphabet.

Definition 4 (Configuration). *Let $\Pi = (\mathcal{S}, \mu, M_1, M_2, \ldots, M_n)$ be a P system as in Definition 3 where $\mathcal{S} = (V, K, D, A, \mathcal{F}, P)$ and $M_i = (w_i, l_i)$, for all $1 \leq i \leq n$. A* configuration (of Π) *is a multiset $C \in \bar{V}$. The* initial configuration (of Π)*, denoted by C_0, is the multiset $u_1\, u_2 \ldots u_n$ such that, for all $1 \leq i \leq n$, u_i is the i-version of w_i.*

Notice that, with the notions introduced in this section so far, we have essentially reduced a P system of degree $n \geq 1$ to an equivalent one of degree 1 where the objects have an index specifying the membrane which they are assigned to in

the original system. Thus, the behavior of such a system can be defined as being a multiset rewriting system where the rules are selected according to a certain strategy depending on the indexes assigned to the objects. In fact, according to Definition 4, a configuration of a P system is just a multiset of objects and the membrane rules are usual multiset rewriting rules.

To this aim, we need first to introduce the concepts of i-irreducibility and the concept of (C, k, q)-consistency.

Definition 5 (i-irreducibility). *Let $\Pi = (\mathcal{S}, \mu, M_1, M_2, \ldots, M_n)$ be a P system as in Definition 3 where $\mathcal{S} = (V, K, D, A, \mathcal{F}, P)$, and let β be a finite multiset over $\bar{V}$. Let MR be the set of membrane rules in Π. Given $1 \leq i \leq n$, we say that β is* i-irreducible *if, for all $u_j\, v_i \rightarrow u'_j\, v'_i \in MP$ with $v_i \neq \lambda$, we have $\beta \not\supseteq u_j\, v_i$.*

Thus, given a P system of degree $n \geq 1$, for all $1 \leq i \leq n$, a multiset over the indexed alphabet is i-irreducible if there are no more rules which can be applied to the objects with index i. In other words, if we interpret these objects as being the content of membrane i, this means that there are no more rules that can be applied to the objects placed inside membrane i.

Definition 6 ((C, k, q)-consistency). *Let $\Pi = (\mathcal{S}, \mu, M_1, M_2, \ldots, M_n)$ be a P system as in Definition 3 where $\mathcal{S} = (V, K, D, A, \mathcal{F}, P)$, and let R be a multiset of rules in Π. Let $C \in \bar{V}^*$ be a configuration of Π, and let $k, q \neq 0$ be positive integers with $k \leq n$. We say that R is* (C, k, q)-consistent *if there exists $\{i_1, i_2, \ldots, i_g\} \subseteq \{1, 2, \ldots, n\}$ with $g \leq k$ such that R can be written as a collection of rules $R_{i_1}, R_{i_2}, \ldots, R_{i_g}$ with R_{i_h} a multiset of rules for membrane i_h, and*

1. *if $g < k$, then, for all $i \in (\{1, 2, \ldots, n\} \setminus \{i_1, i_2, \ldots, i_g\})$, C is i-irreducible;*
2. *for all $i \in \{i_1, i_2, \ldots, i_k\}$, if $R_i = u_j^1\, v_i^1 \rightarrow z_j^1\, w_i^1, \ldots, u_j^p\, v_i^p \rightarrow z_j^p\, w_i^p$, then we have $C = x\, u_j^1\, v_i^1 \ldots u_j^p\, v_i^p$, for some $x \in \bar{V}^*$;*
3. *for all $i \in \{i_1, i_2, \ldots, i_k\}$, $|R_i| \leq q$, and if $R_i = u_j^1\, v_i^1 \rightarrow z_j^1\, w_i^1, \ldots, u_j^p\, v_i^p \rightarrow z_j^p\, w_i^p$ with $p < q$, then $C = x\, u_j^1\, v_i^1 \ldots u_j^p\, v_i^p$, for some i-irreducible $x \in \bar{V}^*$.*

The notion of (C, k, q)-consistency precisely characterizes the multisets of rules which can be applied to a given configuration C in accordance to the parallelism of type (k, q). In fact, such a multiset of rules must contain a multiset of rules for at most k distinct membranes; if there are not k membranes that can evolve by means of some rules, then a smaller but maximal number of membranes must be selected (Condition 1 of Definition 6). The rules contained in the selected multiset must be applicable to the objects currently contained inside each membrane (Condition 2 of Definition 6). Moreover, for each membrane, at most q rules must be selected; if inside some membrane there are less than q rules that can be applied, then all of them must be applied (Condition 3 of Definition 6).

Therefore, in order to perform a (k, q)-parallel step in a given P system, it is necessary to first select a multiset of rules R to be applied to the current configuration C such that R is (C, k, q)-consistent. In this respect, we assume to have defined an *algorithm to select programs and membranes* $\mathcal{A}_{k,q}$ such that,

given a configuration of a P system Π and its set of programs, returns a multiset of rules which is (C, k, q) consistent. In all the previous sections, this selection has been defined as being non-deterministic but, in general, one may identify other strategies which, in particular, should take into account the attributes associated with the rules. Approaches in this direction are considered in [2], [21], [22] where strategies for the selection of the rules are defined which depend on a notion of rate of application of the rules, or on certain probabilities associated with the rules. In the next section, we present one such strategy where the rules to be applied in the next step are selected depending on a particular distribution of probabilities computed step by step.

Here, we define the notion of a (k, q)-parallel step of computation by assuming a generic algorithm for the selection of the rules.

Definition 7 ((k, q)-parallel step). *Let $\Pi = (\mathcal{S}, \mu, M_1, M_2, \ldots, M_n)$ be a P system as in Definition 3 where $\mathcal{S} = (V, K, D, A, \mathcal{F}, P)$, and let C_1, C_2 be configurations of Π. Let $\mathcal{A}_{k,q}$ be an algorithm for the selection of the rules which is able to return a (C_1, k, q)-consistent multiset of rules in Π. We say that C_2 can be obtained from C_1 in a* (k, q)-parallel step*, denoted by $C_1 \Rightarrow_{\Pi}^{(k,q)} C_2$, if there exists a multiset R of rules in Π such that:*

1. $\mathcal{A}_{k,q}(C_1, P) = R$;
2. $R = u_{j_1} v_{i_1} \to z_{j_1} w_{i_1}, \ldots, u_{j_p} v_{i_p} \to z_{j_p} w_{i_p}$, *for some* $p > 0$;
3. $C_1 = x\, u_{j_1} v_{i_1} \ldots u_{j_p} v_{i_p}$ *and* $C_2 = x\, z_{j_1} w_{i_1} \ldots z_{j_p} w_{i_p}$, *for some* $x \in \bar{V}^*$.

If that is the case, then we write $C_1 \Rightarrow_{\Pi}^{(k,q)} C_2$.

Thus, a (k, q)-parallel step in a P system consists in the parallel application of a (C, k, q)-consistent multiset of rules to a certain configuration C. The multiset of rules to be applied is supposed to be returned by a particular algorithm to select membranes and programs, and this has to be done before every step depending on the current configuration of the system.

Then, we introduce the notion of sequence of (k, q)-parallel steps and (k, q)-parallel execution of a P system.

Definition 8 (sequence of (k, q)-parallel steps). *Let $\Pi = (\mathcal{S}, \mu, M_1, M_2, \ldots, M_n)$ be a P system as in Definition 3. A* sequence of (k, q)-parallel steps in Π *is a sequence σ such that*

$$\sigma = C_1, C_2, \ldots, C_h$$

where, for all $1 \leq i \leq h$, C_i is a configuration of Π, and, if $i \neq h$, then $C_i \Rightarrow_{\Pi}^{(k,q)} C_{i+1}$. If that is the case, we say that σ is a sequence of (k, q)-parallel steps in Π that starts from C_1 *and that C_h is* obtained from C_1 in $h - 1$ steps*; we also write $C_1 \Rightarrow_{\Pi}^{(k,q),h} C_h$.*

Definition 9 ((k, q)-parallel execution). *Let $\Pi = (\mathcal{S}, \mu, M_1, M_2, \ldots, M_n)$ be a P system as in Definition 3. A* (k, q)-parallel execution of Π *is a sequence of (k, q)-parallel steps in Π which starts from the initial configuration of Π.*

Thus, we have characterized the behavior of P systems operating according to a bounded parallelism where the number of membranes and the number of rules which can be used in every step are overall bounded by some given constants.

Remark 2. From a computational point of view, the introduction of bounded parallelism in membrane systems does not affect the fundamental universality results concerning the computational power of different variants of P systems, such as P systems with catalysts, with symport/antiport, with boundary rules, etc. In fact, it is easy to see that, in all those cases, the simulation of counter machines is achieved by means of P systems where the number of rules applied in parallel in each step is actually overall bounded (e.g., see [7]). On the other hand, it is shown in [8], [10] that P systems with catalysts operating in sequential mode and P systems with symport/antiport operating in sequential mode (i.e., with parallelism of type $(1, 1)$) are strictly less powerful than their corresponding parallel versions. Moreover, one can also notice that, whenever k is equal to the number of membranes in the system, our notion of parallelism of type (k, q) coincides with the notion of q-**Max-Parallelism** introduced in [7].

4 An Algorithm to Select Membranes and Programs

We present an algorithm to select membranes and programs for P systems operating with parallelism of type $(1, 1)$ (i.e., in sequential mode) where the next membrane to evolve and the next rule to be applied inside this membrane is randomly selected according to a certain distribution of probabilities. However, with respect to Definition 3, the algorithm is here defined only for a restricted model of P systems where rules are all of the forms:

$$u\,[\,] \to [\,v\,], [\,v\,] \to u[\,], [\,v\,] \to [\,v'\,] \tag{2}$$

that is, there is a distinction between transformation rules and communication rules, communication is only unidirectional, and there is no interaction between the inside and the outside of a membrane.

Our strategy for selecting membranes and programs is based on Gillespie's algorithm [12]. This algorithm [12] provides an exact method for the stochastic simulation of systems of bio-chemical reactions; the validity of the method is rigorously proved and it has been already successfully used to simulate various biochemical processes [17]. As well as this, Gillespie's algorithm is used in the implementation of stochastic π-calculus [4] and in its application to the modeling of biological systems [23].

We follow a similar approach to associate a stochastic behavior to membrane systems by considering P systems where each rule has associated a real constant which defines its rate of application and which is used to compute the probability of the rule to be applied in the next step in the same way as in Gillespie's algorithm. More precisely, we consider a class of P systems where, with respect to Definition 3, the set of values is the set of non-negative real numbers denoted by $\mathbb{R}_0^+$, each programs contains a rule like (2), a real constant as an attribute,

and a function to compute a probability depending on the value of this constant. For short, such a P system is called PPR (i.e., a P systems with Probabilities associated with the Rules).

In order to compute the probability values, we use, for all the programs, the function ϕ such that $\phi : \mathbb{R}_0^+ \times V^* \times V^* \longrightarrow \mathbb{R}_0^+$ with:

$$\phi(k, u, \alpha) = k \cdot \prod_{a \in alph(u)} \frac{|\alpha|_a!}{|u|_a! \cdot (|\alpha|_a - |u|_a)!} \tag{3}$$

for all $k \geq 0$, $u, \alpha \in V^*$ and $u \sqsubseteq \alpha$; $\phi(k, u, \alpha) = 0$ for all $k \geq 0$, $u, \alpha \in V^*$ and $u \not\sqsubseteq \alpha$.

That is, given a rule $[\, v\,] \rightarrow u[\,]$, or a rule $[\, v\,] \rightarrow [\, v'\,]$ with an associated attribute k, and given a multiset $\alpha \sqsupseteq v$, expression (3) returns the number of different ways of choosing $|v|_a$ occurrences of a from the multiset α, for all a such that $|a| \geq 0$. In particular, the multiset α is supposed to be the multiset of objects placed inside the membrane where the rule is going to be used. In a similar way, given a rule $u\,[\,] \rightarrow [\, v\,]$ with attribute k, expression (3) is used to compute a probability value for this rule by considering the multiset u and the multiset of objects placed in the outside region.

Remark 3. The function given by expression (3) is already used in [22] to compute probability values for rules. However, this is done in the context of a different algorithm to select rules and programs which is not directly related to Gillespie's algorithm.

Next, we provide a formal definition for the notion of a PPR.

Definition 10 (PPR). *A PPR of degree $n \geq 1$ is a construct*

$$\Pi = (V, K, \mathbb{R}_0^+, A, \phi, P, \mu, M_1, M_2, \ldots, M_n)$$

where:

- $(V, K, \mathbb{R}_0^+, A, \phi, P)$ *is a P specification with ϕ the function given by expression (3), and all the programs in P having the form $\langle r, k, \phi \rangle_l$ for r a rule like (2), and $k \in A$;*
- *μ, and $M_1, M_2, \ldots, M_n$ are as in Definition 3.*

Remark 4. The function ϕ is used to compute the probabilities associated with the rules in a slightly different form with respect to the type of functions considered in Definition 1: only two multisets instead of four are used by the function ϕ. This is because we are restricted to rules of the forms $u\,[\,] \rightarrow [\, v\,]$, $[\, v\,] \rightarrow u[\,]$, $[\, v\,] \rightarrow [\, v'\,]$ containing only one multiset on the left side. However, our approach could be easily generalized to the case of rules of the form $u\,[\, v\,] \rightarrow u'\,[\, v'\,]$ with $u, v \neq \lambda$ by considering a function ϕ' such that

$$\phi'(k, u, v, w_{out}, w_{in}) = \phi(k, u, w_{out}) \cdot \phi(1, v, w_{in})$$

where w_{out}, w_{in} denote the multisets of objects placed respectively inside and outside the membrane where the rule is going to be applied.

Let Π be a PPR, and let C be a configuration of Π. For all $1 \leq i \leq n$, we define the multiset $O_i \in V^*$ as being the multiset of objects such that $|O_i|_a = |C|_{a_i}$, for all $a \in V$ (i.e., O_i is the multiset of objects contained inside membrane i in the configuration C).

We associate to membrane i, with $1 \leq i \leq n$, a set TR_i containing all the triples:

- $(t, v_i \rightarrow u_j, p_t)$, with $\langle [\, v \,] \rightarrow u[\,], k, \phi \rangle_{l_i}$, l_i the label of membrane i, $j = upper(\mu, i)$, and $p_t = \phi(k, v, O_i)$;
- $(t', v_i \rightarrow v'_i, p_{t'})$, with $\langle [\, v \,] \rightarrow [\, v' \,], k, \phi \rangle_{l_i}$, l_i the label of membrane i, and $p_{t'} = \phi(k, v, O_i)$;
- $(t'', u_i \rightarrow v_j, p_{t''})$, with $\langle u [\,] \rightarrow [\, v \,], k_h, \phi \rangle_{l_j}$, $i = upper(\mu, j)$, l_j the label of membrane j, and $p_{t''} = \phi(k, u, O_i)$.

Thus, for each membrane i, the set TR_i is supposed to contain all the rules that can be used inside membrane i with these rules having associated a corresponding value of probability. In particular, if a certain rule is not applicable inside membrane i, then the probability of this rule to be applied turns to be equal to 0. Moreover, notice that rules which send a multiset inside a certain membrane are considered as rules to be used inside the surrounding region.

The following algorithm is then defined to select membranes and programs for P systems with parallelism of type $(1, 1)$.

First, for each membrane i, we compute the index of the next program to be used inside membrane i and its waiting time by using the classical Gillespie's algorithm:

1. calculate $a_0 = \sum p_j$, for all $(j, r, p_j) \in TR_i$;
2. generate two random numbers r_1 and r_2 uniformly distributed over the unit interval $(0, 1)$;
3. calculate the waiting time for the next reaction as $\tau_i = \dfrac{1}{a_0}\, ln(\dfrac{1}{r_1})$;
4. take the index j, of the program such that $\displaystyle\sum_{k=1}^{j-1} p_k < r_2 a_0 \leq \sum_{k=1}^{j} p_k$;
5. return the triple (τ_i, j, i).

Notice that the larger the real constant associated with a rule and the number of occurrences of the objects placed on the left-side of the rule inside a membrane are, the greater the chance that the rule will be applied in the next step of the simulation. There is no constant time-step in the simulation. The time-step is determined in every iteration and it takes different values depending on the configuration of the system.

Next, a step of application of the rules is simulated by using the following procedure:

- **Initialization**
  - set time of the simulation $t = 0$;
  - for each membrane i in μ compute a triple (τ_i, j, i) by using the procedure described above; construct a list containing all such triples;
  - sort the list of triple (τ_i, j, i) according to τ_i;

- **Iteration**
  - extract the first triple, (τ_m, j, m) from the list;
  - set time of the simulation $t = t + \tau_m$;
  - update the waiting time for the rest of the triples in the list by subtracting τ_m;
  - apply the rule contained in the program j only once changing the number of objects in the membranes affected by the application of the rule;
  - for each membrane m' affected by the application of the rule remove the corresponding triple $(\tau'_{m'}, j', m')$ from the list;
  - for each membrane m' affected by the application of the rule j re-run the Gillespie algorithm for the new context in m' to obtain $(\tau''_{m'}, j'', m')$, the next program j'', to be used inside membrane m' and its waiting time $\tau''_{m'}$;
  - add the new triples $(\tau''_{m'}, r'', m')$ to the list and sort this list according to each waiting time and iterate the process.
- **Termination**
  - Terminate simulation when time of the simulation t reaches or exceeds a preset maximal time of simulation, or no more rules can be applied to the objects left inside the membranes.

Therefore, in this approach, it is the waiting time computed by the Gillespie's algorithm to be used to select the membrane which is allowed to evolve in the next step of computation. Specifically, in each step, the membrane associated to the rule with the same minimal waiting time is selected to evolve by means of this rule. If there are more than one rule with the same waiting time, then we assume one of them to be randomly selected to be used in the next step.

Moreover, since the application of a rule can affect more than one membrane at the same time (e.g., some objects may be moved from one place to another), we need to reconsider a new rule for each one of these membranes by taking into account the new distribution of objects inside them.

Remark 5. The use of a variable time-unit for each step does not affect the semantics of our model; in each step, a single rule at a time is applied inside a specific membrane. This means the behavior of the systems is still synchronous although each application of a rule has associated a different time-unit. In fact, the waiting time is mainly used as a parameter necessary to determine the rule to be applied in the next step of computation.

Remark 6. The current algorithm brings some improvements with respect to the notion of step introduced in Definition 7. In fact, in the iteration phase, we need not to recompute all the probabilities associated with each program applicable inside each membrane, but we can do that only for those membranes which are actually affected by the last application of a program. That is so because the value of the probabilities associated with the other rules remain unchanged.

Remark 7. The use of the waiting time parameters leads to selecting a membrane using the minimum waiting time principle. Getting rid of this parameter will lead

to a variant of this algorithm that is associated to an $(n, 1)-$parallel behavior of the system, where n is the total number of membranes. Indeed, in this case there is no way to distinguish between membranes and all of them will be selected.

5 A Case-Study: Bacterial Quorum Sensing

We present an application of membrane systems to the modeling of quorum sensing in bacteria (QS, for short).

The QS mechanism is a communication strategy based on diffusible signals which kick-in under high cellular density. Bacteria use this mechanism to obtain a population-wide coordination of infection, invasion, and evasion of a host's defence. We refer to [13], [14], [28] for further details about the biology of QS. Moreover, a comprehensive bibliography of QS-related research can be found at the web page [30] maintained by the Nottingham Quorum Sensing Group.

QS bacteria produce and release chemical signal molecules, called *autoinducers*, whose external concentration increases as a function of increasing cell-population density. Bacteria detect the accumulation of a minimal threshold stimulatory concentration of these autoinducers and alter their gene expression, and therefore their behavior in response to the variation of the concentration of autoinducers. Using these signal-response systems, bacteria synchronize particular behaviors on a population-wide scale and thus function as multicellular organisms.

The first described quorum-sensing system is that of the bioluminescent marine bacterium *Vibrio fischeri*, and it is considered the basic paradigm for quorum sensing in most (gram-negative) bacteria [18]. *Vibrio fischeri* colonize the light organ of the Hawaiian squid *Euprymna scolopes*. In this organ, the bacteria grow to high cell density and induce the expression of genes for bioluminescence. The squid uses the light provided by the bacteria for counter-illumination to mask its shadow and avoid predation. The bacteria benefit because the light organ is rich in nutrients and allow proliferation in numbers unachievable in seawater. Two proteins, named LuxI and LuxR, control the expression of the luciferase operon (luxICDABE) required for light production. LuxI is the autoinducer synthase, which produces the autoinducer 3OC6-homoserine lactone (OHHL, for short), and LuxR acts as a receptor for these autoinducers. OHHL freely diffuses in and out of the cell and increases in concentration in correspondence of the increasing of the cell density. When this concentration reaches a critical threshold, OHHL binds to LuxR and this complex activates the transcription of the operon encoding luciferase. As well as this, the LuxR-OHHL complex also induces the expression of luxI because it is encoded in the luciferase operon. This regulatory configuration floods the environment with the signal. This creates a positive feedback loop that causes the entire population to switch into "quorum-sensing mode", and produce light; in this case, it is also said that the population is *quorated*.

QS systems have then been identified in other bacterial populations, for instance, *Pseudomonas aeruginosa*, *Vibrio harveyi*, and *Bacillus subtillis*, where

the existence of quorum-sensing networks relying on multiple signalling circuits acting synergistically has also been observed.

5.1 A P System Model of QS

A P system model for the QS system of *Vibrio fischeri* is here defined where a colony of such bacteria is represented by means of a membrane structure consisting of a number of elementary membranes, each one of them representing a bacterium, included in an unique membrane (the skin) representing a common shared environment. In particular, each membrane will contain a set of programs modeling the QS regulatory circuits responsible for the production of light.

To this aim, we use: the symbol $OHHL$ to denote the *autoinducer*, the symbol $LuxR$ to denote the receptor for the autoinducer $OHHL$, the symbol $LuxR\text{-}OHHL$ to denote the complex formed by the binding of the autoinducer $OHHL$ to the receptor $LuxR$, the symbol $LuxBox$ to denote the luciferase operon in its down-regulated state (i.e., when it is not active for the production of light), and the symbol $LuxBox\text{-}LuxR\text{-}OHHL$ to denote the luciferase operon in its up-regulated state (i.e., when it is active for the production of light). Then, we define the following P signature for QS in *Vibrio fischeri*.

$$BS(A) = (V, K, \mathbb{R}_0^+, \phi, A, P)$$

where $A = \{k_1, k_2, k_4, k_3, k_5, k_6, k_7, k_8, k_9, k_{10}, k_{11}, k_{12}, k_{13}, k_{14}\}$ is a set of real constants, $V = \{OHHL, LuxR, LuxR\text{-}OHHL, LuxBox, LuxBox\text{-}LuxR\text{-}OHHL\}$, $K = \{e, b\}$, ϕ is the function given by expression 2, and P is finite set containing all the following programs:

- $\langle [\, LuxBox \,] \to [\, LuxBox\, LuxR \,], k_1, \phi \rangle_b$,
  $\langle [\, LuxBox \,] \to [\, LuxBox\, OHHL \,], k_2, \phi \rangle_b$
  (at low cell density the autoinducer $OHHL$ and the receptor $LuxR$ are produced at a basal rate);
- $\langle [\, OHHL\, LuxR \,] \to [\, LuxR\text{-}OHHL \,], k_3, \phi \rangle_b$,
  $\langle [\, LuxR\text{-}OHHL \,] \to [\, OHHL\, LuxR \,], k_4, \phi \rangle_b$
  (the autoinducer $OHHL$ and the receptor $LuxR$ bind together to form the complex $LuxR\text{-}OHHL$ which, in turn, dissociates in its components);
- $\langle [\, LuxR\text{-}OHHL\, LuxBox \,] \to [\, LuxBox\text{-}LuxR\text{-}OHHL \,], k_5, \phi \rangle_b$,
  $\langle [\, LuxBox\text{-}LuxR\text{-}OHHL \,] \to [\, LuxR\text{-}OHHL\, LuxBox \,], k_6, \phi \rangle_b$
  (the complex $LuxR\text{-}OHHL$ binds to the region of DNA responsible for the production of light; such a complex can also dissociate from that region by returning the luciferase operon to a down-regulated state);
- $\langle [\, LuxBox\text{-}LuxR\text{-}OHHL \,] \to [\, LuxBox\text{-}LuxR\text{-}OHHL\, OHHL \,], k_7, \phi \rangle_b$,
  $\langle [\, LuxBox\text{-}LuxR\text{-}OHHL \,] \to [\, LuxBox\text{-}LuxR\text{-}OHHL\, LuxR \,], k_8, \phi \rangle_b$
  (the binding of the complex to the corresponding region of DNA produces an increase in the production of the autoinducer $OHHL$ and in the production of the receptor $LuxR$);

- $\langle [\, OHHL \,] \rightarrow OHHL\, [\,], k_9, \phi \rangle_b$
  (the autoinducer $OHHL$ freely diffuses outside the bacterium and accumulates in the environment);
- $\langle [\, OHHL \,] \rightarrow [\,], k_{10}, \phi \rangle_b$,
  $\langle [\, LuxR \,] \rightarrow [\,], k_{11}, \phi \rangle_b$,
  $\langle [\, LuxR\text{-}OHHL \,] \rightarrow [\,], k_{11}, \phi \rangle_b$
  (the autoinducer $OHHL$, the receptor $LuxR$ and the complex $LuxR$-$OHHL$ undergo a process of degradation inside the bacterium);
- $\langle OHHL\, [\,] \rightarrow [\, OHHL \,], k_{12}, \phi \rangle_b$
  (the autoinducer $OHHL$ diffuse back from the environment into the bacterium);
- $\langle [\, OHHL \,] \rightarrow [\,], k_{13}, \phi \rangle_e$
  (the autoinducer $OHHL$ is degraded in the environment).

Thus, we have identified 14 rules which model the main transformations involved in the QS system of *Vibrio fischeri*. Notice that the signature BS is parametric with respect to the particular constants associated with the rules.

Next, we define a parametric PPR system $\Pi(n, A)$ to represent a colony of $n \geq 1$ bacteria interacting by means of the QS system described by the aforementioned rules. Specifically, we have

$$\Pi(n, A) = (BS(A), \mu(n), M_1, \ldots, M_n, M_{n+1})$$

where:

- $A = \{k_1, k_2, k_4, k_3, k_5, k_6, k_7, k_8, k_9, k_{10}, k_{11}, k_{12}, k_{13}, k_{14}\}$;
- $\mu(n) = [_{n+1}\, [_1\,]_1 \ldots [_n\,]_n\,]_{n+1}$;
- $M_i = (LuxBox, b)$, for all $1 \leq i \leq n$;
- $M_{n+1} = (\lambda, e)$.

Thus, in the initial configuration, we assume all the bacteria in the colony to contain only one occurrence of the object $LuxBox$ representing the portion of DNA responsible for the production of the autoinducer $OHHL$ and the receptor $LuxR$; the environment is instead supposed to be initially empty. Moreover, notice that, by having the notion of P specification, we can represent an arbitrary large colony in a very compact way by avoiding repeating the same set of programs for every membrane in the system.

Simulation results have been presented under different formalisms and show the same behavior of the colony.

6 Discussion

As we have seen, there is a growing interest in using P systems for modeling biological systems. This often requires the introduction into the membrane system model of some extra features especially when the quantitative aspects characterizing the "reality" of the biological phenomenon to be modeled are considered.

Here we have addressed these issues by specifically introducing the notion of a program consisting of a rule with a finite set of attributes and a function from a given set (Definition 1). We have shown how attributes and functions can be used to define P system models for bio-chemical systems consisting of a number of bio-chemical reactions distributed across various compartments of the system. A precise strategy for the application of the rules has also been defined for this class of P systems which makes possible to associate a stochastic behavior to such P systems. Our approach is based on the well-known Gillespie's algorithm and it is developed alongside the work done in [2], [21], [22] where alternative strategies for the application of the rules are defined.

Acknowledgements

The first author's research is supported by NWO, Organisation for Scientific Research of The Netherlands, project 635.100.006 "VIEWS".

The second and fourth authors are supported by Ministerio de Ciencia y Tecnología of Spain, by *Plan Nacional de I+D+I* (TIN2005-09345-C04-01), cofinanced by FEDER funds, by Junta de Andalucía, by project of Excellence TIC 581, and by a FPU fellowship from the Ministerio de Ciencia y Tecnología of Spain.

References

1. Bernardini, F., Gheorghe, M. (2004). Population P Systems. *J. UCS* **10**,(5), 509–539.
2. Bianco, L., Fontana, F., Manca, V. (2006). P Systems with Reaction Maps. *International Journal of Foundations of Computer Science*, **17**, (1), 27–48.
3. Calder, M., Vyshemirsky, V., Gilbert, D, Orton, R (2006). Analysis of Signalling Pathways using Continuous Time Markov Chains. *Transactions on Computational Systems Biology*, to appear.
4. Cardelli, L., Philips, A.(2004). A Correct Abstract Machine for the Stochastic Pi-calculus. *Electronical Notes in Theoretical Computer Science*, to appear.
5. Ciobanu G., Andrei, O., Lucanu D. (2006). Structural Operational Semantics of P Systems, *WMC6, LNCS* **3850**, 31–48.
6. Collado-Vides, J. (1992). Grammatical Models of the Regulation of Gene Expression, *Proc. of National Academy of Science*, **89**, 9405–9409.
7. Dang, Z., Ibarra, O.H., Li, C., Gaoyan, X. (2006). Decidability of Model-Checking P Systems. *Journal of Automata, Languages and Combinatorics,*, to appear.
8. Dang, Z., Ibarra, O.H. (2005). On One-membrane P systems Operating in Sequential Mode. *Int. J. Found. Comput. Sci.* **16**, (5), 867–881.
9. Duan, Z., Holcombe, M., Bell, A. (2000). A Logic for Biological System, *Biosystems*, **53**, 93–155.
10. Freund, R. (2004). Asynchronous P systems and P systems working in Sequential Mode, *WMC5, LNCS* **3365**, 36–62.
11. Gillespie, D.T. (1976). A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions. *J Comput Physics*, **22**, 403–434.

12. Gillespie, D.T. (1977). Exact Stochastic Simulation of Coupled Chemical Reactions. *The Journal of Physical Chemistry*, **81**, (25), 2340–2361.
13. Hardie, K.R., Williams, P., Winzer, K.(2002). Bacterial cell-to-cell communication: sorry, can't talk now, gone to lunch. *Current Opinion in Microbiology*, **5**, 216–222.
14. Fargerströn, T., James, G., James, S., Kjelleberg, S., Nilsson, P. (2000). Luminescence Control in the Marine Bacterium Vibrio fischeri: An Analysis of the Dynamics of lux Regulation. *J. Mol. Biol.* **296**, 1127–1137.
15. Kam, N., Cohen, I.R., Harel, D. (2001). The Immune System as a Reactive Systems: Modelling T Cell Activation with Statecharts, *The Weizmann Institute of Science*, Israel.
16. Matsuno, H., Doi, A., Nagasaki, M., Miyano, S. Hybrid (2000). Petri Net Representation of Gene Regulatory Network, *Pacific Symposium on Biocompting*, World Scientific, 338–349.
17. Meng, T.C., Somani S., Dhar, P. (2004). Modelling and Simulation of Biological Systems with Stochasticity. *In Silico Biology*, **4**, (0024), 137–158.
18. Nealson, K.H., Hastings, J.W. (1979). Bacterial Bioluminescence: Its Control and Ecological Significance, *Microbiology Review*, **43**, 496–518.
19. Păun, Gh. (2000). Computing with Membranes, *Journal of Computer and System Sciences*, **61**, (1), 108 – 143.
20. Păun, Gh. (2002). *Membrane Computing. An Introduction*, Springer, Berlin.
21. Pérez-Jiménez, M.J., Romero-Campero, F.J. (2006). P Systems, a New Computationl Modelling Tool for Systems Biology, *Transactions on Computational Systems Biology VI, LNBI*, **4220**, 176-197.
22. Pescini, D., Besozzi, D., Mauri, G., Zandron, C. (2006). Dynamical probabilistic P systems, *International Journal of Foundations of Computer Science*, **17**, (1), 183–195.
23. Priami, C., Regev, A., Shapiro, E., Silverman, W. (2001). Application of a Stochastic Name-Passing Calculus to Representation and Simulation of Molecular Processes, *Information Processing Letters* **80**, 25–31.
24. Regev, A., Shapiro, E. (2004) The π-calculus as an abstraction for biomolecular systems. In Gabriel Ciobanu and Grzegorz Rozenberg, editors, *Modelling in Molecular Biology*, Springer, 219–266.
25. Segel, I.H. (1976). *Biochemical Calculations: How to Solve Mathematical Problems in General Biochemistry*, John Wiley and Sons, 2nd edition.
26. Till, J.E., McCulloch, F. Siminovitch, L. (1964). A Stochastic Model of Stem Cell Proliferation based on the Growth of Spleen Colony-Forming Cells, *Proc. National Academy of Science USA*, **51**, 117–128.
27. Walker, D., Holcombe, M., Southgate, J., McNeil, S., Smalwood, R. (2004). The Epitheliome: Agent-Based Modelling of The Social Behaviour of Cells, *Biosystems*, **76**, (1–3), 89–100.
28. Waters, C.M., Bassler, B.L. (2005). Quorum Sensing: Cell-to-Cell Communication in Bacteria. *Annu. Rev. Cell. Dev. Biol.* **21**, 319–346.
29. The P Systems Web Site: `http://psystems.disco.unimib.it`
30. Nottingham Quorum Sensing Web Site `http://www.nottingham.ac.uk/quorum/`

Synchrony and Asynchrony in Membrane Systems

Jetty Kleijn[1] and Maciej Koutny[2]

[1] LIACS, Leiden University
P.O.Box 9512, NL-2300 RA Leiden, The Netherlands
kleijn@liacs.nl
[2] School of Computing Science, University of Newcastle
Newcastle upon Tyne, NE1 7RU, United Kingdom
maciej.koutny@ncl.ac.uk

Abstract. We consider synchrony and asynchrony in the behavior of various models of membrane systems, which may differ in the way individual reactions are defined as well as in the way multisets of these reactions can be executed in a single computational step. We concentrate on the properties of ongoing computations, including the unbounded ones. Our focus is on the properties of system states involved in such computations as well as on concurrency and causality relationships between executed reactions. This should be contrasted with the approach which investigates different notions of 'results' produced through halting computations of membrane systems. As a formal behavioral model we use Petri nets and their processes which are very well suited to capture the notion of an execution in a concurrent context. We continue our earlier work reported in [15], where a systematic and structural link has been established between a basic class of membrane systems and Petri nets. Here, we look at some natural extensions of this basic class of membrane systems and investigate the ways in which they can be represented within the behavioral model provided by Petri nets.

1 Introduction

Inspired by the way living cells are divided by membranes into compartments where biochemical reactions may take place, *membrane systems* (also known as *P systems*) have become a prominent new computational model [1,21,23,24]. In a nutshell, a reaction transforms multisets of molecules (or objects) present in the compartment into new molecules, possibly transferring some to neighboring compartments and the environment. Consequently, all aspects of the dynamic behavior of membrane systems are determined by the *reaction* or *evolution rules* in each compartment and by the way in which these rules occur. The resulting transformations (or computation steps) take place starting from an initial configuration (a distribution of objects). Furthermore, a notion of a successful (or halting) computation with its output is defined [23,24]. Different types of membrane systems have been considered, depending on the form of the rules and how they are applied, and on input/output definitions. In fact, studies in the field of

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 66–85, 2006.

membrane systems are often concerned with investigating the possible outcomes of the computations, i.e., the computational power of the various models. The aim of our work, however, is different in that we are interested in describing what is actually going on *during* an execution of a membrane system; alternatively, one might say that we are interested in *computations* rather than *computability*. Thus, we focus on possible system states (configurations) occurring in ongoing computations as well as on the concurrency and causality relationships between executed reaction rules. This emphasis on possible behaviors (runs) rather than input/output relations, further implies that *all* possible computations need to be considered, including non-successful and still ongoing (even unbounded) ones (which are also relevant from a biological/cell point of view).

There are basically two distinguishing features of any model of membrane systems when one is interested in the structural properties of their executions rather than in successful computations or the results produced. The first is the definition of individual reactions; in the simplest case, a reaction is supposed only to consume and produce multisets of molecules, but in more elaborate models, its execution can, e.g., be conditional or affect the structure of the cell. The second is the degree of synchrony present in a single computation step; in the extreme case, commonly considered in the theory of membrane systems, in a single step the system is transformed by a maximally concurrent execution of reaction rules (no more rules in whatever compartment could have been applied). In this paper, we will consider different kinds of synchrony as well as different types of reaction rules, and we will indicate how *Petri nets* (see, e.g., [10,28]) can be used to capture the structural properties of the computations of varying models of membrane systems.

Petri nets are bipartite directed graphs consisting of two kinds of nodes, called *places* and *transitions*. Places indicate the local availability of resources (represented by so-called tokens) and thus can be used to represent objects in specific compartments. Transitions are actions which can occur depending on local conditions related to the availability of resources and they can be used to represent reaction rules associated with compartments. When a transition occurs it consumes resources from its input places and produces items in its output places, thus mimicking the effect of a reaction rule (see Figure 2). Since multiset calculus is basic for membrane systems and also for computing the token distribution in Petri nets [7], some connections between the two models were already established including interpretations of reaction rules of membrane systems using Petri net transitions (e.g., [9,27]). Petri nets are a *fundamental* modeling tool for relations between occurrences of actions, moreover providing both a language and a method for behavioral analysis through so-called processes to formalize the concept of a concurrent run and with a corresponding theory of labeled partial orders. Note that models such as process algebras do not yield themselves as easily to the modeling of membrane systems since the structure of the latter is relatively simple, and the main advantage of the former, viz. compositionality in system specification and execution, is not needed.

This paper builds on previous work [15,16], where it has been demonstrated that a structural relationship between Petri nets and membrane systems can be established at the system level. A formal translation has been given from a basic class of membrane systems into a class of Petri nets. The direct correspondence of Petri net transitions together with their input and output places to evolution rules is the key property which makes the translation suitable for dealing with structural aspects of the behavior of membrane systems. It implies that the causality and concurrency relations between applications of reaction rules are preserved in the relationships between occurrences of the corresponding transitions. Thus also the synchrony in computation steps corresponds to potentially simultaneously occurring transitions. As shown in [15], in case the membrane system evolves in a synchronous fashion (i.e., with a maximally concurrent execution of reaction rules in each computation step), its computations are faithfully reflected in the maximally concurrent step sequence semantics of its Petri net. In *Place/Transition nets with localities* (or PTL-nets), the specific class of Petri nets introduced in [15], each transition moreover belongs to a location, similar to the distribution of the reaction rules over the compartments in a membrane system. Since locality aspects of the resources consumed and produced by transitions is explicitly supported by their underlying graph structure, this locality information is not relevant for the maximal concurrency semantics of a Petri net. However, transitions with associated localities can be used to restrict synchrony to certain locations: in each step, and for each locality actively involved in that step, as many transitions belonging to this locality as possible are executed. Thus the PTL-net model and its *locally* maximal concurrency semantics facilitate the investigation of membrane systems working under the natural assumption that synchrony is restricted to individual compartments. Observe that this semantics leads to a more general model of membrane systems: maximal concurrency can be studied in the framework of PTL-nets with only one locality. PTL-nets with the locally maximal concurrency semantics provide a formal framework for the modeling and analysis of so-called 'globally asynchronous locally synchronous', or GALS, systems. Other examples of such systems occur, e.g., in hardware design (see [8,30]) when computations take place in synchronous clusters with asynchronous data exchange.

In general, a step sequence semantics for Petri nets provides important insights into concurrency aspects of a system when executed. Such semantics, however, are based on ordered sequences of steps which may obscure the true causal relationships between occurrences of transitions since not all ordering is a consequence of causality. Still information on causal relationships is often highly relevant for system design and analysis. As was recognized a long time ago (see [20]), Petri nets support a formal approach where this information is readily available. Runs (as given, e.g., by step sequences) are unfolded (unraveling their steps) into structures which explicitly represent causality and concurrency. For this purpose, labeled occurrence nets, called *processes* are used (see, e.g., [4,5,12,29]). The standard process semantics of Place/Transition nets (based on arbitrary steps) does not work in the PTL case due to lack of information on

potential executability of transitions relevant for the local maximality of executed steps. To cope with this problem, in [15] the occurrence nets generated by PTL-nets are adapted leading to the notion of barb-processes formally defined and investigated in [16]. In [15,16] however, only a very basic class of membrane systems is considered with simple evolution rules and evolving in a (locally) synchronous fashion. In this paper, we will attempt to establish a similar set-up for other existing, more sophisticated, variants and extensions of membrane systems. For each of these variants, we intend to define a suitable (extension of the) PTL-net model with a proper semantics. Obviously, we aim at retaining the direct correspondence between (occurrences of) transitions and (application of) evolution rules in order to guarantee that (local) synchrony and asynchrony in the membrane systems have corresponding interpretations in the PTL-net. Note that this work is a preliminary investigation, and full technical details are left to forthcoming papers.

2 Preliminaries

In this paper, a multiset (over a set X) is a function $\mathsf{m} : X \to \mathbb{N}$. By $\mathbb{N}^X$ we denote the set of multisets over X. For two multisets m and m' over X, we denote $\mathsf{m} \leq \mathsf{m}'$ if $\mathsf{m}(x) \leq \mathsf{m}'(x)$ for all $x \in X$. Moreover, a subset of X may be viewed through its characteristic function as a multiset over X, and for a multiset m we denote $x \in \mathsf{m}$ if $\mathsf{m}(x) \geq 1$. Multiset m over X is finite if there are finitely many $x \in X$ such that $\mathsf{m}(x) \geq 1$; the cardinality of m is then defined as $|\mathsf{m}| \stackrel{\text{df}}{=} \sum_{x \in X} \mathsf{m}(x)$. The sum of two multisets m and m' over X is given by $(\mathsf{m}+\mathsf{m}')(x) \stackrel{\text{df}}{=} \mathsf{m}(x)+\mathsf{m}'(x)$, and the difference by $(\mathsf{m}-\mathsf{m}')(x) \stackrel{\text{df}}{=} \max\{0, \mathsf{m}(x)-\mathsf{m}'(x)\}$, i.e., as a total function extending set difference. The multiplication of m by a natural number n is given by $(n \cdot \mathsf{m})(x) \stackrel{\text{df}}{=} n \cdot \mathsf{m}(x)$. Moreover, any finite sum $\mathsf{m}_1 + \cdots + \mathsf{m}_k$ will also be denoted as $\sum_{i \in \{1,\ldots,k\}} \mathsf{m}_i$.

2.1 Basic Membrane Systems

We first consider the most basic definition of membrane systems. A *(basic) membrane system* (of degree $m \geq 1$) [21,24] is a construct

$$\Pi \stackrel{\text{df}}{=} (V, \mu, w_1^0, \ldots, w_m^0, R_1, \ldots, R_m)\,,$$

where:

- V is a finite *alphabet* consisting of (names of) objects;
- μ is a *membrane structure* given by a rooted tree with m nodes, representing the membranes — we assume that the nodes are given as the integers $1, \ldots, m$, and $(i, j) \in \mu$ will mean that there is an edge from i (parent) to j (child) in the tree of μ;
- each w_i^0 is a multiset of objects initially associated with membrane i;

– each R_i is a finite set of *reaction rules* (or *evolution rules*) r associated with membrane i, of the form $lhs^r \to rhs^r$, where lhs^r — the left hand side of r — is a non-empty multiset over V, and rhs^r — its right hand side — is a possibly empty multiset over

$$V \cup \{a_{out} \mid a \in V\} \cup \{a_{in_j} \mid a \in V \wedge (i,j) \in \mu\}\,.$$

The nodes of a membrane structure represent membranes which in their turn determine the compartments: node j represents membrane m_j which defines c_j as the compartment enclosed by m_j and in-between m_j and its children if any. In the above, symbols a_{in_j} represent objects a that will be sent to (the compartment defined by) the child node j and a_{out} stands for an a that will be sent out to the parent's compartment. We assume that no evolution rule r associated with the root of the membrane structure uses any a_{out} in rhs^r.

A membrane system Π as above evolves from configuration to configuration as a consequence of the application of (multisets of) evolution rules in each compartment. Formally, a *configuration* is a tuple $C \stackrel{\text{df}}{=} (w_1, \ldots, w_m)$ where each w_i is a multiset of object names; we define a *vector multi-rule* $\boldsymbol{R}$ as an element of $\mathbb{N}^{R_1} \times \cdots \times \mathbb{N}^{R_m}$. Given a vector multi-rule $\boldsymbol{R} = (\widehat{R}_1, \ldots, \widehat{R}_m)$, we use as additional notation $lhs_i = \sum_{r \in R_i} \widehat{R}_i(r) \cdot lhs^r$ for the multiset of all objects in the left hand sides of the rules in $\widehat{R}_i$ and, similarly, $rhs_i = \sum_{r \in R_i} \widehat{R}_i(r) \cdot rhs^r$ is the multiset of all — possibly indexed — objects in the right hand sides.

We now come to a point where we need to make precise the execution semantics of the basic membrane system model. As we already mentioned, it can be defined in a number of ways, depending on the balance between *synchrony* and *asynchrony* in the allowed behaviors. We will consider four kinds of execution semantics that have been investigated in the area of membrane systems, i.e., the common *maximal* parallelism, the *locally maximal* parallelism from [15,16], *minimal* parallelism [11], and *free* parallelism [25].

First, we consider free parallelism by which any combination of reaction rules can be executed as a synchronous step provided that enough resources are available. More precisely, configuration $C = (w_1, \ldots, w_m)$ *free-evolves* into configuration $C' = (w'_1, \ldots, w'_m)$ by a vector multi-rule $\boldsymbol{R} = (\widehat{R}_1, \ldots, \widehat{R}_m)$, or $C \stackrel{\boldsymbol{R}}{\Longrightarrow}_{free} C'$, if for every $1 \leq i \leq m$, $lhs_i \leq w_i$ and, for each object $a \in V$,

$$w'_i(a) = w_i(a) - lhs_i(a) + rhs_i(a) + rhs_{parent(i)}(a_{in_i}) + \sum_{(i,j) \in \mu} rhs_j(a_{out})\ ,$$

where $parent(i)$ is the father membrane of i unless i is the root in which case $parent(i)$ is undefined and $rhs_{parent(i)}(a_{in_i})$ is omitted. Note that any j in the last term must be a child of i. By the first condition, the configuration C has in each membrane i enough occurrences of objects for the application of the multiset of evolution rules $\widehat{R}_i$, and the second condition describes the effect of the application of the rules in $\boldsymbol{R}$.

The other three execution semantics can be seen as restrictions of the free parallelism paradigm. Given $C \stackrel{\boldsymbol{R}}{\Longrightarrow}_{free} C'$ as above, we say that C:

- *min-evolves* into C' (or $C \stackrel{\boldsymbol{R}}{\Longrightarrow}_{min} C'$) if $|\widehat{R}_1| + \cdots + |\widehat{R}_m| = 1$;
- *max-evolves* into C' (or $C \stackrel{\boldsymbol{R}}{\Longrightarrow}_{max} C'$) if there is no i and rule r in R_i such that $lhs^r + lhs_i \leq w_i$; and
- *lmax-evolves* into C' (or $C \stackrel{\boldsymbol{R}}{\Longrightarrow}_{lmax} C'$) if there is no i and rule r in R_i such that $lhs^r + lhs_i \leq w_i$ and $|\widehat{R}_i| \geq 1$.

A *free/min/max/lmax-computation* of Π is then defined to be a sequence of free/min/max/lmax-evolutions starting from $C_0 \stackrel{\text{df}}{=} (w_1^0, \ldots, w_m^0)$, the initial configuration of Π.

2.2 Petri Nets with Localities

We now recall the key notions of the standard Petri net model. A *PT-net* is a tuple $N \stackrel{\text{df}}{=} (P, T, W, M_0)$ such that P and T are finite disjoint sets and $W : (T \times P) \cup (P \times T) \to \mathbb{N}$ and $M_0 : P \to \mathbb{N}$ are multisets. The elements of P and T are respectively the *places* and *transitions* of N, W is the *weight function* of N, and M_0 is its *initial* marking. In diagrams, places are drawn as circles, and transitions as rectangles. If $W(x, y) \geq 1$ for some $(x, y) \in (T \times P) \cup (P \times T)$, then (x, y) is an *arc* leading from x to y. As usual, arcs are annotated with their weight if this is 2 or more. We assume that, for every $t \in T$, there is a place p such that $W(p, t) \geq 1$. Places represent local states (resources), while markings are multisets of places (depicted by the corresponding number of tokens, small black dots, in each place) representing the global states of a PT-net. Transitions represent actions which may occur at a given marking and then lead to a new marking (the weight function specifies what resources are consumed and produced during the execution of such actions).

The *pre-* and *post-multiset* of a transition $t \in T$ are multisets of places given, for all $p \in P$, by: $\text{PRE}_N(t)(p) \stackrel{\text{df}}{=} W(p, t)$ and $\text{POST}_N(t)(p) \stackrel{\text{df}}{=} W(t, p)$. Both notations extend to multisets of transitions U:

$$\text{PRE}_N(U) \stackrel{\text{df}}{=} \sum_{t \in U} U(t) \cdot \text{PRE}_N(t) \qquad \text{and} \qquad \text{POST}_N(U) \stackrel{\text{df}}{=} \sum_{t \in U} U(t) \cdot \text{POST}_N(t) \; .$$

In order to represent the compartmentalization of membrane systems, one can add the notion of located transitions. In the proposed way of specifying locality for the transitions in a PT-net, each transition belongs to a fixed unique locality. The exact mechanism for achieving this is to introduce a partition of the set of all transitions, using a locality mapping $\mathfrak{D}$. Intuitively, two transitions for which $\mathfrak{D}$ returns the same value will be co-located. Consider the PTL-net depicted in Figure 1. Transitions `a` and `c` are assigned one locality, whereas transitions `t` and `u` are assigned another locality. This PTL-net is a model of a producer/consumer system which reflects the view that producers operate away (at location 1) from consumers (location 2).

A *PT-net with localities* (or PTL-net) is a tuple $NL \stackrel{\text{df}}{=} (P, T, W, M_0, \mathfrak{D})$, where $\text{UND}(NL) \stackrel{\text{df}}{=} (P, T, W, M_0)$ is the *underlying* PT-net and $\mathfrak{D} : T \to \mathbb{N}$ is a *location mapping* for the transition set T. In the diagrams of PTL-nets, transitions are

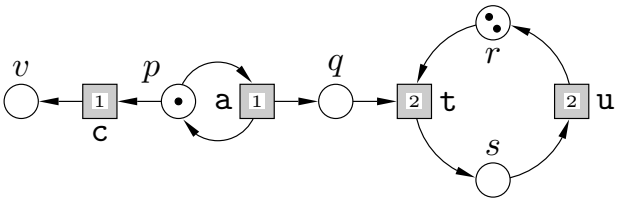


Fig. 1. PTL-net of the one-producer/two-consumers system

shaded rectangles with the locality being shown in the middle. Note that $\mathfrak{D}$ is merely a labeling of transitions, it is not meant as a renaming (as used later for occurrence nets).

We now can introduce execution semantics for the PTL-net which closely reflect the different degrees of synchrony in basic membrane systems.

A *step* is a multiset of transitions, $U : T \to \mathbb{N}$. It is *free-enabled* at a marking M (or $M[U\rangle_{free}$) if $M \geq \text{PRE}_N(U)$. Thus, in order for U to be free-enabled at M, for each place p, the number of tokens in p under M should at least be equal to the total number of tokens that are needed as an input to U, respecting the weights of the input arcs. We further say that U is:

- *min-enabled* at M (or $M[U\rangle_{min}$) if $|U| = 1$;
- *max-enabled* at M (or $M[U\rangle_{max}$) if there is no transition t such that we have $M[U + \{t\}\rangle_{free}$; and
- *lmax-enabled* at M (or $M[U\rangle_{lmax}$) if there is no transition t such that we have $M[U+\{t\}\rangle_{free}$ and $\mathfrak{D}(t) \in \mathfrak{D}(U)$. (Note that localities are only relevant for lmax-enabledness.)

Let $\mathfrak{m} \in \{free, min, max, lmax\}$ be a mode of execution. If U is $\mathfrak{m}$-enabled at M, then it can be $\mathfrak{m}$-*executed* leading to the marking $M' \stackrel{\text{df}}{=} M - \text{PRE}_N(U) + \text{POST}_N(U)$. This means that the execution of U 'consumes' from each place p exactly $W(p,t)$ tokens for each occurrence of a transition $t \in U$ that has p as an input place, and 'produces' in each place p exactly $W(t,p)$ tokens for each occurrence of a transition $t \in U$ with p as an output place. If the $\mathfrak{m}$-execution of U leads from M to M' we write $M[U\rangle_{\mathfrak{m}}M'$. A sequence $\sigma = U_1 \ldots U_n$ of non-empty steps is an $\mathfrak{m}$-*step sequence* (from the initial marking M_0) if there are markings $M_1, \ldots, M_n$ of N satisfying $M_{i-1}[U_i\rangle_{\mathfrak{m}}M_i$ for every $i \leq n$. Such a σ is called an $\mathfrak{m}$-step sequence from M_0 to M_n, and M_n is an $\mathfrak{m}$-*reachable* marking.

2.3 From Basic Membrane Systems to PTL-Nets

We now recall the details of the translation from the basic model of membrane system to PTL-nets introduced in [15]. Let $\Pi = (V, \mu, w_1^0, \ldots, w_m^0, R_1, \ldots, R_m)$ be a membrane system of degree m. Then the corresponding PTL-net is $NL_\Pi \stackrel{\text{df}}{=} (P, T, W, M_0, \mathfrak{D})$ where the various components are defined thus:

- $P \stackrel{\text{df}}{=} V \times \{1, \ldots, m\}$ and $T \stackrel{\text{df}}{=} T_1 \cup \ldots \cup T_m$, where each T_i consists of distinct transitions t_i^r for every evolution rule $r \in R_i$;

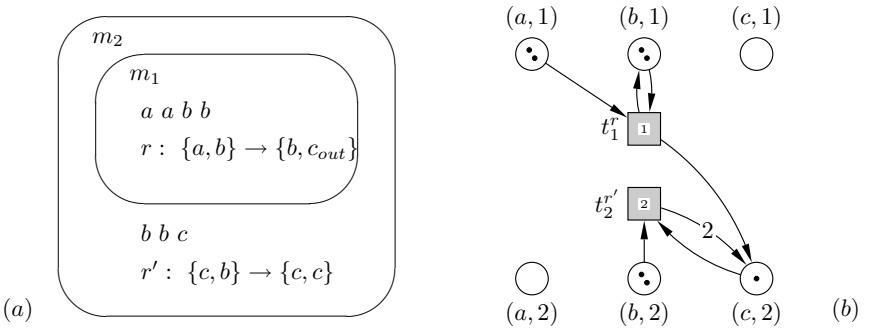


Fig. 2. A membrane system (a); and the corresponding PTL-net (b)

– for every place $p = (a, j) \in P$ and every transition $t = t_i^r \in T$,

$$W(p,t) \stackrel{\text{df}}{=} \begin{cases} lhs^r(a) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \qquad W(t,p) \stackrel{\text{df}}{=} \begin{cases} rhs^r(a) & \text{if } i = j \\ rhs^r(a_{out}) & \text{if } (j,i) \in \mu \\ rhs^r(a_{in_j}) & \text{if } (i,j) \in \mu \\ 0 & \text{otherwise} \end{cases}$$

– for every place $p = (a, j) \in P$, its initial marking is $M_0(p) \stackrel{\text{df}}{=} w_j^0(a)$.
– for every transition $t = t_i^r \in T$, its locality is $\mathfrak{D}(t) \stackrel{\text{df}}{=} i$.

An example is the membrane system depicted in Figure 2(a). It consists of two nested membranes (m_1 and m_2), two rules (rule r associated with m_1, and rule r' associated with m_2; m_1 is the child and m_2 is the root in the membrane structure), and three symbols denoting molecules (a, b, and c). Initially, the compartment c_1 inside m_1 contains two copies of both a and b, and c_2, in-between the two membranes, contains two copies of b and a single copy of c. To model this membrane system as a PTL-net, we introduce a place (x, j) for each kind of molecule x and compartment c_j. For each rule r associated with m_i we introduce a separate transition t_i^r with locality i. If the transformation described by a rule r of membrane m_i consumes k copies of molecule x from compartment c_j, then we introduce a k weighted arc from place (x, j) to transition t_i^r, and similarly for molecules produced by transformations. Finally, assuming that, initially, compartment c_j contained n copies of molecule x, we introduce n tokens into place (x, j). The resulting PTL-net is depicted in Figure 2(b).

Let $C = (w_1, \ldots, w_m)$ be a configuration of Π. Then the corresponding marking $\phi(C)$ of NL_Π is given by $\phi(C)(a, i) \stackrel{\text{df}}{=} w_i(a)$, for every place (a, i) of NL_Π. Similarly, for any vector multi-rule $\boldsymbol{R} = (\widehat{R}_1, \ldots, \widehat{R}_m)$ of Π, we define a multiset $\psi(\boldsymbol{R})$ of transitions of NL_Π such that $\psi(\boldsymbol{R})(t_i^r) \stackrel{\text{df}}{=} \widehat{R}_i(r)$ for every $t_i^r \in T$. Note that ϕ is a bijection from the configurations of Π to the markings of NL_Π, and ψ is a bijection from vector multi-rules of Π to steps of NL_Π.

We now can formulate a fundamental property concerning the relationship between the dynamics of the basic membrane system Π and that of the

corresponding PTL-net. Let $\mathtt{m} \in \{free, min, max, lmax\}$ be a mode of execution of membrane systems. Then: $C \stackrel{\boldsymbol{R}}{\Longrightarrow}_{\mathtt{m}} C'$ *if and only if* $\phi(C)\ [\psi(\boldsymbol{R})\rangle_{\mathtt{m}}\ \phi(C')$. Since the initial configuration of Π corresponds through ϕ to the initial marking of NL_Π, the above immediately implies that the $\mathtt{m}$-computations of Π coincide with the $\mathtt{m}$-step sequences of the PTL net NL_Π.

2.4 Causality and Concurrency in System Behavior

The step sequences defined by the four different modes of execution of PTL-nets provide important insights into the concurrency aspects of the thus defined behavior. They are, however, still sequential in nature in the sense that steps occur ordered thus obscuring the true causal relationships between the occurrences of transitions. Yet Petri nets can easily support a formal approach where this information is readily available by unfolding behaviors (step sequences) into structures which allow an explicit representation of causality, conflict, and concurrency (see [20]). A well-established way of developing such a semantics is based on a class of acyclic Petri nets, called *occurrence nets* [29]. What one essentially tries to achieve is to trace the changes of markings due to transitions being executed along some legal behavior of the original PT-net, and in doing so record which resources were consumed and produced.

Recall that for free parallelism, localities are not relevant. Looking at the (free-)step sequence $\sigma = \{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$ of the PTL-net in Figure 1, it is not immediate that transition $\mathtt{u}$ could have occurred before the second occurrence of transition $\mathtt{a}$ or, in other words, that the former is not causally dependent on the latter.

Figure 3 illustrates the idea of how to *unfold* σ. The initial stage represents only the initial marking which includes two separate (labeled) *conditions* (this is how places are called in occurrence nets), each representing an initial token in place r. Executing step $\{\mathtt{a}\}$ consumes the p-condition, creates an $\mathtt{a}$-labeled event (this is how transitions are called in occurrence nets), as well as two new conditions: a p-condition and a q-condition. An important point is to notice that we create a fresh p-condition rather than a loop back to the initial one since we want to distinguish between different occurrences of a token in the same place; as a result the occurrence net being constructed will be an acyclic graph. Another important point is that the environment of the generated $\mathtt{a}$-event corresponds exactly to the environment of transition $\mathtt{a}$; namely, it consumes a p-token and creates a p-token and a q-token. After that, executing step $\{\mathtt{t},\mathtt{a}\}$ consists in consuming three conditions and creating two events and three fresh conditions, and similarly for the last step $\{\mathtt{u},\mathtt{t}\}$. As a final result, we obtain an acyclic net labeled with places and transitions of the original PT-net; it is called a *process* of the original PT-net. The underlying, unlabeled, net is referred to as an occurrence net. Note that in these nets each condition has at most one incoming arc and at most one outgoing arc (is *non-branching*) as it is caused by at most one event and is available as a resource to at most one event. In particular, there are no choices (conflicts) between events as these have been resolved during the

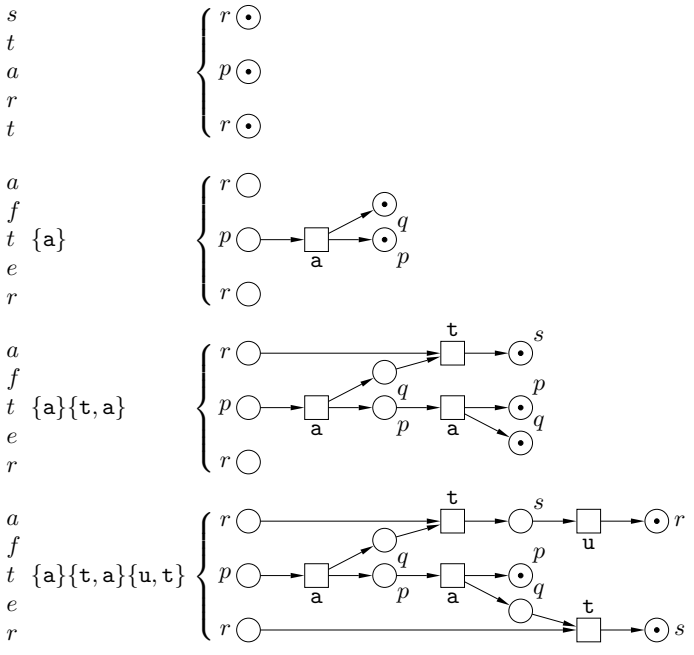


Fig. 3. Constructing a process corresponding to {a}{t, a}{u, t}

run. The occurrence net has a default initial marking consisting of a token in each of the conditions without an incoming arc.

It is now possible to look both at the structure of the occurrence net and at the (labelled) executions which are possible from its default initial marking, making some important observations (see, e.g., [13] and the semantical framework outlined there) relating to:

- *Causality.* The causality relationships among the executed transitions can be read-off by following directed paths between the events; for example in Figure 3, the lower (second) t-event is caused by both a-events, while the upper (first) one is caused only by the leftmost a-event.
- *Concurrency.* Events for which there is no directed path from one to another can be thought of as concurrent; in Figure 3, the second a-event and the u-event are concurrent (not causally related); and so are the two t-events.
- *Reachability.* Any maximal set of conditions for which there is no directed path from one condition to another corresponds to a (free-)reachable marking of the original PT-net.
- *Representation.* The step sequence on basis of which the process was created can be executed from the initial default marking in the occurrence net. So the original behavior has been retained. In Figure 3, there are several different (labeled) free-step sequences (e.g., {a}{t}{a, u}{t}) that can be executed by the occurrence net defined by $\sigma = \{a\}\{t, a\}\{u, t\}$, including σ itself.

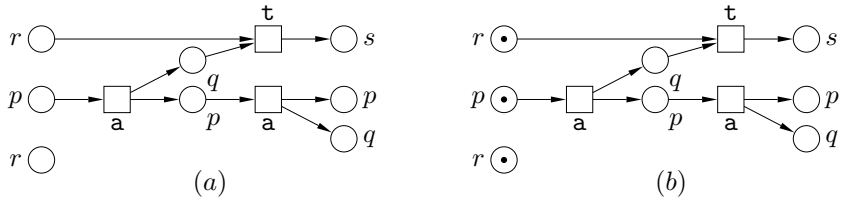


Fig. 4. Process corresponding to {a}{t, a} (a); and its default initial marking (b)

– *Executability.* Any (labeled) step sequence of the occurrence net (from the default initial marking to the default final marking (consisting of tokens placed in each of the conditions without an outgoing arc) is a legal step sequence of the original PT-net. (The step sequence {a}{t}{a, u}{t} in Figure 3 is a free-step sequence of the net in Figure 1.)

From the point of view of causality and concurrency, the minimal parallelism semantics is almost exactly the same as in the case of free parallelism. The only difference is that executability is formulated with respect to step sequences where each step is a singleton, rather than a general finite multiset of transitions.

To deal with locally maximal parallelism, as a first attempt, we simply adopt the unfolding strategy as in the case of free parallelism. We only ensure that the step sequence consists of lmax-steps. Moreover, we preserve the localities of the transitions in the events created while constructing the occurrence net. Figure 4 shows the result for the PTL-net of Figure 1 and the lmax-step sequence {a}{t, a}. Then we need an argument that the resulting process is what one would want to take for further analyzes. In particular, one would want to retain executability as in the previous construction.

In the case of our example, we can execute the occurrence net and conclude that under the locally maximal parallelism it admits the step sequence {a}{a}{t} which is not a legal lmax-step sequence of the PTL-net of Figure 1 since after {a}{a}, two occurrences of t are enabled. Thus, in general it would be too hasty to accept the standard unfolding routine as satisfactory since information on (additional) enabledness may be lost. Consider further the PTL-net in Figure 5(a) and its lmax-step sequence {t, u, v}{w, z}. Proceeding as in the case of free parallelism, we obtain an occurrence net as shown in Figure 5(b). Now the problem is that it has an lmax-step sequence from the default initial marking which through its labels corresponds to {u, v}{t, z}{w}. This, however, is not an lmax-step sequence of the original PTL-net. An intuitive reason is that the standard unfolding 'forgets' that transition x was enabled at the stage where transition w was selected. Then, delaying the execution of the w-event, creates a situation where the executed step (though lmax-enabled within the occurrence net) does not correspond to an lmax-step in the PTL-net.

To cope with this problem, [16] added to occurrence nets special *barb-events*, depicted as darkly shaded rectangles. Barb-events are not labeled with transition names and are not meant to be executed; rather, they are used in the

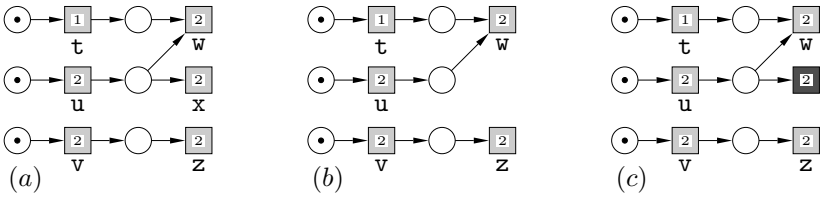


Fig. 5. PTL-net (a); an occurrence net constructed from $\{\mathtt{t},\mathtt{u},\mathtt{v}\}\{\mathtt{w},\mathtt{z}\}$ (b); and a barbed process (c)

calculation of the enabled sets of events. Such modified labeled occurrence nets are called *barbed* processes. Rather than providing a full formal definition of how barb-events are added during the unfolding procedure, which can be found in [16,17], we only mention here that it is based on checking for the existence of locally newly enabled transitions not (yet) included in the executed scenario, e.g., since another co-located transition was selected. Figure 5(c) illustrates the modified construction for the nets in Figure 5(a,b). After executing $\{\mathtt{u},\mathtt{v}\}$, it is now impossible to select $\{\mathtt{t},\mathtt{z}\}$ since there is a record in the form of the barb-event that such a step would not be maximal in the locality to which transition $\{\mathtt{z}\}$ belongs. The only way of continuing is to execute $\{\mathtt{t}\}$ and after that $\{\mathtt{z},\mathtt{w}\}$, generating a legal lmax-step sequence $\{\mathtt{u},\mathtt{v}\}\{\mathtt{t}\}\{\mathtt{z},\mathtt{w}\}$.

The maximal parallelism semantics of a PTL-net coincides with the locally maximal parallelism semantics of this PTL-net after changing it so that all transitions are mapped to the same locality.

3 Extensions Expressible Within PTL-Nets

In the previous section we outlined the way in which the basic membrane systems can be translated into PTL-nets, and their behavioural properties investigated using processes of the latter. In the rest of the paper, we will change focus and investigate what happens if more sophisticated types of reaction rules are allowed. For the sake of simplicity, we will assume from now on that the membrane systems and PTL-nets are executed according to the free parallelism paradigm (notice that the level of synchrony present in executions is orthogonal to the way individual reaction rules are specified).

We start by considering extensions for which the PTL-net semantics can be used without any, or with only slight, modifications. These extensions have been discussed in [23,2] and additional references will be provided throughout the text. Note that each extension is motivated by some natural phenomenon in the area of biological systems.

Catalysts. In this variant, a subset Cat of objects, called *catalysts*, is distinguished and each reaction rule r is of the form $lhs^r \rightarrow rhs^r$ with either no catalysts involved at all or with $lhs^r(c) = rhs^r(c) = 1$, for exactly one $c \in Cat$, and $lhs^r(c') = rhs^r(c') = 0$, for all other catalysts c'. In other words, in certain

reaction rules a catalyst has to participate, but it is neither destroyed in the process nor can be created. Clearly, since catalysts can be seen as resources for the reaction rules in which they occur (to be returned after application), these rules can be translated into PTL-transitions in exactly the same way as any other rule. Thus, the translation from Section 2.3 is fully adequate. Similarly, other variants of catalysts, such as m-stable catalysts and mobile catalysts, can also be treated by this basic translation.

Rules creation and consumption. Within the basic model of membrane systems no assumptions are made with respect to the number of times a reaction rule is available for application in a single execution step. Now, it is assumed that reaction rules are finite resources in the same way as the objects located in compartments [3]. More precisely, each configuration has additional information for each membrane about the number of locally available copies of reaction rules. Each rule r is of the form $lhs^r \rightarrow rhs^r/z$, with z a multiset over the set of rules. Rules are executed in the usual manner with respect to the multisets of objects consumed and created. Moreover, if r when executed is associated with membrane i, then a copy of r is consumed from the multiset of rules currently available in i and the multiset z is added to that pool. Note that we may assume that each rule occurs associated with each membrane.

In this case the translation proceeds as in Section 2.3 with two key modifications: (i) for each transition t_i^r corresponding with rule r in association with membrane i, a unique *control place* is added which acts as a counter and indicates the number of copies of r available in the corresponding membrane; (ii) this control place is an additional input place to t_i^r and if r is of the form $lhs^r \rightarrow rhs^r/z$, then t_i^r has, for each control place corresponding with a rule $r' \in z$ associated with i, an additional output place with weight $z(r')$.

Systems with i/o communication. These systems are defined as in Section 2.1, except that in rules r of the form $lhs^r \rightarrow rhs^r$ the right hand side rhs^r is a multiset over $V \cup \{a_{out} \mid a \in V\} \cup \{a_{in} \mid a \in V\}$. The index in of a_{in} means that a copy of object a is to be moved into *any* of the inner membranes of the membrane to which r belongs. Thus every rule represents a set of rules of the original form, each such rule corresponding to a combination of non-deterministic choices of inner membranes for all occurrences of an a_{in}. As a result, the translation has to be lifted to a more abstract level and each reaction rule involving sending objects to inner membranes is translated into a set of transitions with the same pre-multisets, but possibly different post-multisets. For example, if 3 and 7 are two inner membranes for the rule $ab \rightarrow c_{in}d_{in}$, then this rule is translated into four transitions, with the following post-multisets: $\{(c,3),(d,3)\}$, $\{(c,3),(d,7)\}$, $\{(c,7),(d,3)\}$ and $\{(c,7),(d,7)\}$.

Symport/antiport. In a symport/antiport membrane system the rules associated with a membrane i are of one of the forms (x, in) or (y, out) — symport rules — or $(x, in; y, out)$ — antiport rules. Here (x, in) means that the multiset x is moved from the outside of i to its inside and, similarly, (y, out) means that

multiset y goes from the inside of i to its outside. Moreover, $(x, in; y, out)$ means that x and y are moved simultaneously. Note that with the given membrane structure, it must be the case that x moves from the 'location' of the parent of i to i and y in the opposite direction. Consequently, we can again apply the basic translation in case of rules of the first two forms. The third one is somewhat different since it consumes objects from two neighboring compartments. However, its translation is straightforward and what we simply obtain is a transition taking tokens from places corresponding to different compartments of the original membrane system.

Tissue membrane systems. In this case objects are transported through channels rather than membranes. Thus the nested tree-like structure of membranes is replaced by a graph, with its edges representing channels connecting compartments in a completely arbitrary way. Often it is assumed that at most one (symport or antiport) rule associated with a channel is executed at any given moment. Since the actual membrane structure is not relevant for the translation, the first assumption has no effect, and the translation looks as in the case of symport and antiport rules. The second assumption can be addressed by introducing, for each communication channel, a special place marked initially with a single token which is connected by a pair of arcs (pointing in opposite directions) with every transition representing a reaction rule associated with that channel. In this way, there can never occur more than one of these transitions at the same time. As in the case of rules creation and consumption, these additional places are an example of what might be called a 'control structure' which can be used in the Petri net model to implement a specific behavioral aspect of membrane systems.

4 Other Extensions

Although it is possible to use the basic class of PTL-nets to analyze various important classes of membrane systems, not all interesting phenomena can be modeled by using purely the features of PTL-nets.

Promoters. It is now assumed that a reaction rule r can have the form $lhs^r \rightarrow rhs^r|_c$ meaning that c is a promoter object which has to be present for the rule to be executed [6]. It should be stressed that such an object is not a catalyst since catalysts are actively involved in reactions, whereas a single occurrence of c in its role of promoter may enable simultaneously two or more executions of the rule. It turns out that the standard model of PTL-nets is no longer sufficient for the modelling of promoters because arcs between transitions and places indicate consumption and production. We need an extension with (weighted) *activator* arcs [13], represented by arcs with small black dots at the end. We call the extended model PTLA-nets. Activator arcs represent 'tests' for the presence of tokens in places. An activator arc of weight n between place p and transition t implies that the latter can only be executed if the former contains at least n tokens. The resulting marking is calculated in exactly the same way as before, i.e., activator arcs have no effect on the result and are simply ignored.

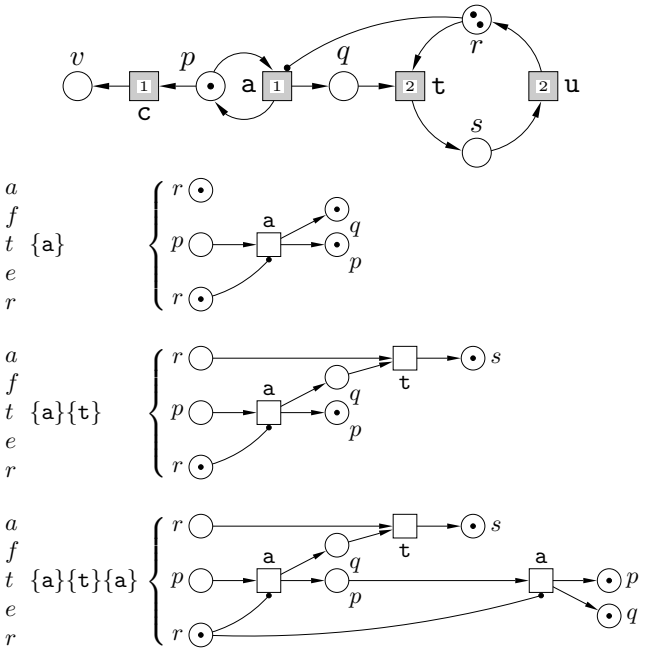


Fig. 6. PTLA-net of the one-producer/two-consumers system with a non-eager producer, and constructing an ao-process corresponding to {a}{t}{a}

The translation of reaction rule $lhs^r \to rhs^r|_c$ associated to membrane i proceeds as the basic translation for $lhs^r \to rhs^r$, and after that an activator arc of weight 1 is added to link the resulting transition with place (c, i). A more elaborate definition of promoters assumes the format $lhs^r \to rhs^r|_u$ where u is a multiset of objects. The translation then proceeds similarly but now a number of activator arcs of weights greater or equal to 1 as described by u are added at the end.

The process semantics of the resulting translations can no longer be captured using the standard process semantics of Petri nets. What we use are *activator processes* (or *ao-processes*) which are occurrence nets with additional (weight 1) activator arcs between events and conditions to test for the presence of tokens in the places corresponding to the conditions. With the distinguishing feature of activator arcs being that they do not consume conditions, there may be several activator arcs adjacent to a single condition (in addition, of course, to the standard directed arcs, and in contrast with the non-branching of conditions with respect to ordinary arcs). Consider, for instance, the net in Figure 6 which models a system where the producer only produces items if there is at least one consumer waiting for them. A possible free-step sequence is {a}{t}{a} and constructing the corresponding ao-process is illustrated in Figure 6. Notice that we have here a condition connected by activator arcs to two different events.

The causality semantics of ao-processes is no longer the same as that of the standard processes. Basically, in the latter causality is based on partial orders

whereas in ao-processes another relationship, called *weak causality*, is needed. It turns out that the standard partial order treatment of causality can be extended to cover its weak variant as well, and the main results and properties can be recovered [13,14].

Inhibitors. Inhibitors are objects the presence of which makes the execution of certain rules impossible. In this case, a reaction rule r can have the form $lhs^r \rightarrow rhs^r|_{\neg c}$ meaning that c is an object which, when present in the compartment, inhibits the execution of this rule [6]. Again, and for the same reason as with promoters, we need to extend PTL-nets, in this case with weighted *inhibitor* arcs [13,26], represented by arcs with small circles at the end. We call the extended model PTLI-nets. The meaning of an inhibitor arc of weight $n \geq 0$ between place p and transition t is that the latter can only be executed if the former contains at most n tokens (thus, if $n = 0$ then the place must be empty of tokens). The resulting marking is calculated in exactly the same way as before, i.e., inhibitor arcs have no effect on the result and are simply ignored.

The translation of reaction rule $lhs^r \rightarrow rhs^r|_{\neg c}$ associated to membrane i proceeds as the basic translation for $lhs^r \rightarrow rhs^r$, and after that an inhibitor arc of weight 0 is added to link the resulting transition with place (c, i). A more elaborate definition of inhibitors assumes the format $lhs^r \rightarrow rhs^r|_{\neg u}$ where u is a set of object symbols. The translation proceeds then similarly but now inhibitor arcs of any weights can be added.

The process semantics of the resulting translations can be captured using the ao-process semantics as in the case of promoters. Consider, for instance, the net in Figure 7 which models a system where the producer can cancel the production of items only if there is no consumer waiting for them. A possible free-step sequence is $\{\mathtt{a}\}\{\mathtt{a},\mathtt{t}\}\{\mathtt{t}\}\{\mathtt{c}\}$ and the construction of the corresponding ao-process is illustrated in Figure 7. Note that activator arcs rather than inhibitor arcs are used to test for the holding of conditions. Two activator arcs are used to represent the test for the presence of two tokens in the place s. This ensures that place r is empty since in the PTLI-net of Figure 7 the total number of tokens in places s and r is always equal to 2. In Petri net terminology places like that are called *complementary* and the modeling of inhibitor arcs in a process semantics is then rather straightforward. In case a complement for a place like r cannot be found, a more elaborate construction can be used to achieve the desired effect [13,14]. Since the causality semantics of PTLI-nets is based on ao-processes, it takes into account weak causality, as for PTLA-nets.

Permeable membranes. The new kind of reaction rule allowed here is $lhs^r \rightarrow rhs^r/\tau$. The special symbol τ indicates that rule r, when executed, causes its associated enclosing membrane to become 'thick' (or non-permeable), and no object can pass through it anymore [22]. To render this feature within the Petri net model, we introduce a special, initially empty, place $perm_i$ associated with the membrane i. A directed arc is added to $perm_i$ from those transitions which correspond to rules which make membrane i thick (thus there may be several transitions which can put tokens into the control place $perm_i$). Then each

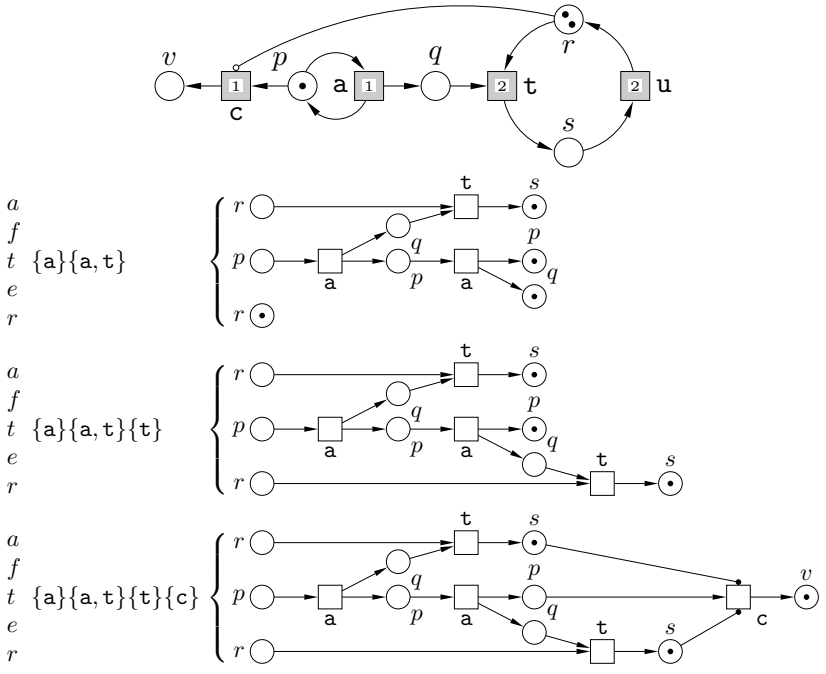


Fig. 7. PTLI-net of the one-producer/two-consumers system with considerate producer, and constructing an ao-process corresponding to {a}{a, t}{t}{c}

transition t which models a reaction rule r' transferring objects through membrane i, is connected with $perm_i$ using a simple inhibitor arc. Hence, as long as no transition which places a token into the control place $perm_i$ is executed, transitions transferring objects through the membrane i are possible. However, once there is at least one token in $perm_i$, transitions corresponding to rules like r' can no longer be executed as no transition can remove tokens from $perm_i$. Hence PTLI-nets with the associated ao-processes are sufficient to model the effect of the non-permeability of membranes.

Dissolving membranes. To dissolve membranes, reaction rules are used of the form $lhs^r \to rhs^r /\delta$ with δ a special symbol indicating that execution of this rule causes its associated enclosing membrane (which may not be the outermost membrane) to dissolve [22,21]. Moreover, all objects present in the compartment are incorporated into the immediately enclosing compartment, and all the rules associated to membrane i are rendered inapplicable. The dissolving of membranes may be modeled by a combination of activator and inhibitor arcs. Take, for example, a membrane system with four membranes, arranged into a line-like tree $1 \to 2 \to 3 \to 4$. Assume further that there are three dissolving rules: r' associated with 2, r'' associated with 3, and r''' associated with 4. Then we add three control places for keeping information about dissolved membranes: $diss_2$,

$diss_3$ and $diss_4$ which are initially empty, and each has an incoming directed arc from the transitions corresponding respectively to r', r'' and r'''.

Suppose now that we need to translate a rule $r : a \to aa$ associated with membrane 2. To achieve the desired effect, we introduce three transitions with locality 2: t_2^r with pre-multiset $\{(a, 2)\}$ and post-multiset $\{(a, 2), (a, 2)\}$; t_3^r with pre-multiset $\{(a, 3)\}$ and post-multiset $\{(a, 3), (a, 3)\}$; and t_4^r with pre-multiset $\{(a, 4)\}$ and post-multiset $\{(a, 4), (a, 4)\}$. Then we add an inhibitor arc between $diss_2$ and each of these three transitions, as well as three activator arcs: between t_3^r and $diss_3$, t_4^r and $diss_3$, t_4^r and $diss_4$. In this way, t_4^r can be executed only if the dissolution rules r'' and r''' have happened, but r' has not. Note that such a translation can properly render even a simultaneous application of multiple instances of the dissolution rules. Process semantics of the resulting Petri net is a combination of those of PTLA-nets and PTLI-nets.

5 Concluding Remarks

A main advantage of the process semantics of Petri nets is that it provides a very compact representation of the (step sequence) behavior of a net. This feature has been exploited in the development of efficient model checking algorithms [19], where issues relating to reachability of certain configurations and termination (or deadlock) of a system can be addressed. Given a process notion for membrane systems obtained via a faithful translation into (a specific kind of) Petri nets, relevant behavioral properties can be efficiently investigated. For example, one can check for the presence of certain molecules (also in specific compartments), by suitably adapting the notion of reachability in an occurrence net. In some cases, it might be important to know whether local computations within compartments (and across the whole system) are independent of each other, and answering this kind of question could amount to checking for the causal links between various events present in processes.

The classical process semantics of Petri nets is based on labeled occurrence which provide a faithful representation of causality and concurrency in the case of asynchronously operating nets (and the corresponding membrane systems). As we pointed out, to deal with semantics involving synchrony (such as locally maximal parallelism) these processes need to be augmented with additional information, resulting in barbed processes.

In this paper, we discussed a number of extensions of basic membrane systems and their transformation into equivalent Petri nets. It turned out that some extensions can be treated with the existing PTL-nets model (perhaps after adding additional control structures to the existing translation). However, other extensions need more expressive processes, and we proposed to augment (barbed) processes with so-called activator arcs (which have already been investigated within the Petri net theory).

There are several possible directions for future work. The first is to complete the development of the theory of barb-processes making it fit into the *semantical framework* of [13]. Also the Petri net semantics for extensions of the basic

class of membrane systems should be further developed leading to suitable process notions and derived causality structures. Another important question is a complete characterization of the state graphs generated by various models of PTL-nets allowing, in particular, to answer the question whether a given state graph could have been generated by a membrane system of a given kind (such a characterization has so far been provided for the class of safe PTL-nets [18]). Last but not least, once a sound notion of behavioral characterization of a membrane system has been provided, one can re-introduce the notion of a successful computation, the result it produces, and the notion of an input-output relation taking into account both non-successful and ongoing computations.

Acknowledgment. We are grateful to Grzegorz Rozenberg for introducing us to the area of membrane systems and many inspiring discussions. This research was supported by the EPSRC project CASINO.

References

1. Membrane systems web page: `http://psystems.disco.unimib.it/`
2. Alhazov, A.: Communication in Membrane Systems with Symbol Objects. PhD Thesis, Rovira i Virgili University, Tarragona, Spain (2006)
3. Arroyo, F., Baranda, A.V., Castellanos, J., Păun, G.: Membrane Computing: The Power of (Rule) Creation. Journal of Universal Computer Science **8** (2002) 369–381
4. Best, E., Devillers, R.: Sequential and Concurrent Behaviour in Petri Net Theory. Theoretical Computer Science **55** (1988) 87–136
5. Best, E., Fernández, C.: Nonsequential Processes. A Petri Net View. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin (1988)
6. Bottoni, P., Martín-Vide, C., Păun, G., Rozenberg, G.: Membrane Systems with Promoters/Inhibitors. Acta Informatica **38** (2002) 695–720
7. Calude, C.S., Păun, G., Rozenberg, G., Salomaa, A. (eds.): Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View. Lecture Notes in Computer Science, Vol. 2235. Springer-Verlag, Berlin (2001)
8. Carloni, L.P., Sangiovanni-Vincentelli, A.L.: A Formal Modelling Framework for Deploying Synchronous Designs on Distributed Architectures. Proc. of First International Workshop on Formal Methods for Globally Asynchronous Locally Synchronous Architectures (2003)
9. Dal Zilio, S., Formenti, E.: On the Dynamics of PB Systems: a Petri Net View. In: Martín-Vide, C., et al. (eds.): WMC 2003. Lecture Notes in Computer Science, Vol. 2933. Springer-Verlag, Berlin (2004) 153–167
10. Desel, J., Reisig, W., Rozenberg, G. (eds.): Lectures on Concurrency and Petri Nets. Lecture Notes in Computer Science, Vol. 3098. Springer-Verlag, Berlin (2004)
11. Freund, R.: Sequential P Systems. Romanian Journal of Information Science and Technology **4** (2001) 77–88
12. Goltz, U., Reisig, W.: The Non-sequential Behaviour of Petri Nets. Information and Control **57** (1983) 125–147
13. Kleijn, H.C.M., Koutny, M.: Process Semantics of General Inhibitor Nets. Information and Computation **190** (2004) 18–69
14. Kleijn, H.C.M., Koutny, M.: Infinite Process Semantics of Inhibitor Nets. In: Donatelli, S., Thiagarajan, P.S. (eds.): ICATPN 2006. Lecture Notes in Computer Science, Vol. 4024. Springer-Verlag, Berlin (2006) 282–301

15. Kleijn, H.C.M., Koutny, M., Rozenberg, G.: Towards a Petri Net Semantics for Membrane Systems. In: Freund, R., et al. (eds.): WMC 2005. Lecture Notes in Computer Science, Vol. 3850. Springer-Verlag, Berlin (2006) 292–309
16. Kleijn, H.C.M., Koutny, M., Rozenberg, G.: Process Semantics for Membrane Systems. To appear in the Journal of Automata, Languages and Combinatorics (2006)
17. Kleijn, H.C.M., Koutny, M., Rozenberg, G.: Processes of Petri Nets with Localities. Report 941, School of Computing Science, University of Newcastle (2006)
18. Koutny, M., Pietkiewicz-Koutny, M.: Transition Systems of Elementary Net Systems with Localities. In: Baier, C., Hermanns, H. (eds.): CONCUR 2006. Lecture Notes in Computer Science, Vol. 4137. Springer-Verlag, Berlin (2006) 173–187
19. McMillan, K.L.: Using Unfoldings to Avoid the State Explosion Problem in the Verification of Asynchronous Circuits. In: von Bochmann, G., Probst, D.K. (eds.): CAV 1992. Lecture Notes in Computer Science, Vol. 663. Springer-Verlag, Berlin (1992) 164–174
20. Nielsen, M., Plotkin, G., Winskel, G.: Petri Nets, Event Structures and Domains, Part I. Theoretical Computer Science **13** (1980) 85–108
21. Păun, G.: Computing with Membranes. Journal of Computer and System Sciences **61** (2000) 108–143
22. Păun, G.: Computing with Membranes – A Variant. International Journal of Foundations of Computer Science **11** (2000) 167–182
23. Păun, G.: Membrane Computing, An Introduction. Springer-Verlag, Berlin (2002)
24. Păun, G., Rozenberg, G.: A Guide to Membrane Computing. Theoretical Computer Science **287** (2002) 73–100
25. Păun, G., Yu, S.: On Synchronization in P Systems. Fundamenta Informaticae **38** (1999) 397–410
26. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice Hall (1981)
27. Qi, Z., You, J., Mao, H.: P Systems and Petri Nets. In: Martín-Vide, C., et al. (eds.): WMC 2003. Lecture Notes in Computer Science, Vol. 2933. Springer-Verlag, Berlin (2004) 286–303
28. Reisig, W., Rozenberg, G. (eds.): Lectures on Petri Nets. Lecture Notes in Computer Science, Vol. 1491 and 1492. Springer-Verlag, Berlin (1998)
29. Rozenberg, G., Engelfriet, J.: Elementary Net Systems. In: [28] (1998) 12–121
30. Stahl, C., Reisig, W., Krstić, M.: Hazard Detection in a GALS Wrapper: a Case Study. In: Desel, J., Watanabe, Y. (eds.): ACSD'05, IEEE Computer Society (2005)

MP Systems Approaches to Biochemical Dynamics: Biological Rhythms and Oscillations

Vincenzo Manca

University of Verona
Department of Computer Science
Strada Le Grazie, 15
37134 Verona, Italy
vincenzo.manca@univr.it

Abstract. Metabolic P systems are a special class of P systems which seem to be adequate for expressing biological phenomena related to metabolism and signaling transduction in biological systems. We give the basic motivation for their introduction and some ideas about their applicability to some basic biological oscillators.

1 Introduction

P systems were introduced as a new computation model, inspired by biology [31,32], where *multisets* and *membranes* are the two main ingredients. The theory of P systems has grown very fast by studying different kinds of evolution rules and strategies. Important mathematical results have been established on the computational power of different kinds of P systems and on their relationships with other computational models [32,39]. The state of a P system is given by the multisets of objects present inside each membrane. The passage from a state to another is produced by the application of rules (a set of rules for each membrane) which act independently in each membrane and, typically, are applied in a *maximally parallel way*. This means that a maximal set of rules which are applicable is chosen and applied in a parallel way.

The P system paradigm has also been used to mathematically model several biomolecular phenomena acting at the cellular level, such as trans-membrane transport and communication [29,30], consumption of energy [16,33] and even more specific biological processes [15,4,36,9].

Early attempts of symbolic descriptions of metabolic processes were initiated by the author, approximately ten years ago [23,24]. In these papers some primitive notions of membrane systems were considered, but the use of logical formulae driving metabolite concentrations made them too general for expressing biological situations in a significant way. The theory of P systems was crucial in two important steps toward a new symbolic model of a metabolic system. A first step was the dynamical perspective in the study of P systems, introduced in [3], where the dynamical patterns of P systems were the main focus of investigation. A second step was the introduction of a molar perspective, borrowed from chemistry, with an abstract notion of "reaction strength" as a parameter able to

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 86–99, 2006.

regulate the cooperation/competition among the rules of P systems [25]. In fact, in a very first approximation, a cell is a membrane system, and its functioning is determined by all the types of molecules inside it, by the amount of molecules of these types, and by the cell compartments where they are located [1]. Therefore, it is of great importance to define a method for computing the evolution of a P system that is directly meaningful with respect to biochemical reactions.

The *Brusselator*, which is a differential model of a chemical oscillator, inspired by the famous Belousov-Zhabotinsky reaction, was modeled in [34,35] by means of multiset rewriting rules. This model suggested us the idea of defining a general algorithmic procedure on P systems which could provide results comparable with the classical differential models, but using different principles. Three main points emerged in this direction: i) population multiset rewriting rules, instead of object rewriting rules, ii) observation (discrete) time, instead of (continuous) reaction time, and iii) a criterion for computing, at each step, the masses of reactants which the rules need for producing their products.

In this perspective, a transformation $AA \rightarrow BC$ is better read in chemical terms, as something which expresses the following prescription: "two *moles* of A produce one mole of B and a mole of C". Here a *mole* is a conventional population unit like *a battalion, a company, a brigade*, which is not conceived in an absolute way, as it happens in the classical chemical setting (1 mole $\approx 6.02 \times 10^{23}$ molecules), but it is relative to the specific system. If we fix the number of objects of a mole, then the dimension of a multiset is expressed, in terms of moles, by a rational number.

When many reactions are working together, a competition among reactions needing the same kinds of reactants is better expressed by a notion of *reaction unit*. For example, a rule such as $AA \rightarrow BC$ should reasonably say that $2m$ objects of type A have to be consumed by the rule, and m objects B plus m objects C have to be produced by it. The crucial point of this discussion is "how has the number m to be calculated in order to reproduce adequately a given biochemical process?" This problem becomes more difficult than it may appear at a first glance. Such a number depends on the relative strength of a rule with respect to the other rules which are competing with it for the same reactants. We call this strength *reactivity*, and in general, it depends on the current state of the system.

MP systems [27] formalize these intuitions by considering P systems with a particular deterministic procedure for computing their evolution. This procedure, called MP Algorithm, shortly MPA [28], aims at capturing the salient chemical mechanisms that are responsible for the dynamics of a wide class of biomolecular processes. We have shown that MP systems effectively model the dynamics of several biochemical processes: the Belousov-Zhabotinsky reaction (Brusselator) [6,8], the Lotka-Volterra dynamics [25,7,6,8,13], a Susceptible-infected-recovered epidemic [6], the Leukocyte selective recruitment in the immune response [15,6], the Protein Kinase C Activation [8], Circadian rhythms [12], and Mitotic cycles [28]. Other phenomena under investigation concern

Cdc25A degradation in tumor processes, an oscillatory circuit that includes Protein Kinases ERK2 and PK [22] and the intercellular communication which occurs in Dictyostelium discoideum. This organism is an amoeba very important in developmental biology, which may switch from unicellular to multicellular stages (isolated and collective phases) by means of a periodic chemical mechanism, similar to hormonal communications in higher organisms [18]. A project, of interest in the search for biological energy sources, intends to apply MP systems to the analysis of specific metabolism occurring in microbial fuel cells. From a more theoretical point of view, interesting relationships were stated between MPA and ODE (ordinary differential equations) [14]. In the section 3, we show the relevance of these theoretical results in relation to an example of biological modeling [17].

In the analysis of MP systems and of their applications an important role is played by MP graphs [28], which we will briefly outline, and which yield an immediate formulation of the structural aspects of MP systems in a style similar to other graphical representations in signal transduction networks and metabolic pathways [20,38].

2 Metabolic P Systems

MP systems are deterministic P systems where i) the state of the system, at each time instant, is given by the amount of matter that is assigned to any (chemical) substance present in the system, and ii) the transition to the next state (after some specified interval of time) is calculated according to some *mass partition strategy*, that is, the available matter of each substance is partitioned among all reactions which need to consume it. The policy of matter partition is regulated at each instant by some real values, called *reactivities*, which represent the strength of any reaction.

The definition we give here of MP systems is similar, but different to those given in our preceding papers on this subject. In the present form it seems more appropriate to the further theoretical and experimental development of these systems, especially in the process of providing models from the data of biological observations.

A *discrete multiset* over an alphabet T is a function from T to the set $\mathbb{N}$ of natural numbers. A *continuous multiset* over an alphabet T is a function from T to the set $\mathbb{R}$ of real numbers. As it is customary in P systems, we will adopt the string notation for discrete multisets. Sometimes it is useful to use the symbol + for concatenation, in order to stress that in multisets concatenation is commutative, that is, when a string denotes a multiset, the order of its symbols is not relevant (see [32] for more details on P systems notations) and, for any string α, we write $X \in \alpha$ for saying that X is a symbol occurring in α. The set Q_T of states over an alphabet T consists of the continuous multisets over T. The passage from discrete to continuous states is motivated by the use of moles for determining the mass associated to each symbol of T.

The notion of MP system we consider here should be better identified by that of *zero level* MP system, because only one membrane is considered.

Definition 1 (MP System). *An MP system is a construct*

$$M = (T, R, F, \nu, \mu, \tau, q_0)$$

in which

- T *is a finite set of symbols;*
- R *is a finite set of rules, i.e., pairs of discrete multisets over* T *(represented, as usual, in the arrow notation);*
- F *is the set of reaction maps, such that* $F = \{f_r \mid r \in R\}$*, where* $f_r : Q_T \to \mathbb{R}$*. Very often the reactivity* $f_r(q)$ *in the state* q *depends only on the mass associated to some of the symbols of* T*. For this reason, it is convenient to introduce a real variable* $x = q(X)$ *to any symbol* $X \in T$*. We write* $f_r(x, y, \ldots)$ *to explicitate the variables* $x, y, \ldots$ *which* f_r *depends on;*
- ν *is a natural number which specifies the value of a (conventional) mole of* M*;*
- μ *is a function which assigns to each* $X \in T$*, the mass* $\mu(X)$ *of a mole of* X*, with respect to some measure unit;*
- τ *is the temporal interval between two consecutive states;*
- q_0*, the initial state of* M*, is an element of* Q_T*.*

The temporal evolution of an MP system M is calculated by means of a *metabolic difference operator* Δ_q, which provides for any state $q \in Q_T$ a function

$$\Delta_q : Q_T \to \mathbb{R}$$

such that, for every $X \in T$, the state following q in the temporal evolution of M is given by $q(X) + \Delta_q(X)$.

Two assumptions are fundamental in the definition of reaction rules and reaction maps used by MPA, which directly relate to the perspective of mass partition strategy adopted for MP systems evolution.

Principle 1 (Inertia). *In any MP system, for every* $X \in T$*, a rule* r_X *is present, which is called inertial rule for the substance* X*, and such that* $X \to X$*. The inertia of* r_X *is the reactivity of an inertial rule* r_X*, in a given state, and indicates the tendency of substance* X *to remain unchanged.*

Principle 2 (Creativity). *Any input rule* r *of type* $\lambda \to X$ *is assumed to be, implicitly, transformed into a rule* $\lambda_r \to \lambda_r X$ *where* λ_r *is a new symbol in* T*, called the input symbol of* r*. This means that a sort of input gate, as a container of a given capacity of* X*, is assumed to feed the system from the outside, at a rate depending on the reactivity of the input rule. This capacity determines the* creativity *of the rule* $\lambda \to X$*, as the maximum value of elements* X *that can enter into the system at each evolution step.*

The value of inertia of each element of T (possibly extended with input symbols), and the value of the creativity of input rules are very important parameters for the evolution of MP systems according to the strategy we are going to define.

In order to define our MP algorithm, which formalizes the intuition given at beginning of Section 2, we use the following notation from [28], that will be adopted in the rest of the paper and it will be always related to a metabolic system $M = (T, R, F, \nu, \mu, \tau, q_0)$.

Definition 2 (MP Notation)

- *Each $r \in R$ is denoted by $r : \alpha_r \to \beta_r$; α_r identifies the multiset of the substrates of r and β_r identifies the multiset of the products of r;*
- *$h_r(X)$ is the number of occurrences of X in α_r;*
- *$g_r(X)$ is the number of occurrences of X in β_r;*
- *$R_\alpha(X) = \{r \in R \,|\, X \in \alpha_r\}$;*
- *$R_\beta(X) = \{r \in R \,|\, X \in \beta_r\}$;*
- *$R(X) = R_\alpha(X) \cup R_\beta(X)$;*
- *$\Pi(\alpha_r) = \prod_{X \in \alpha_r} q(X)^{h_r(X)}$ (by definition, $\Pi(\alpha_r) = 1$ if $\alpha_r = \lambda$).*

We assume that if $\alpha_r = \lambda$, then $\beta_r \in T$, and if $\beta_r = \lambda$ then $\alpha_r \in T$.

Definition 3 (MPA). *The value of the metabolic difference operator Δ_q of an MP system, in a state $q \in Q_T$ and on a symbol $X \in T$, is given by:*

$$\Delta_q(X) = \sum_{r \in R(X)} (g_r(X) - h_r(X)) \cdot u_r(q)$$

where

$$u_r(q) = \min\Big\{ w_{Y,q}(r) \frac{q(Y)}{h_r(Y)} \mid Y \in \alpha_r \Big\}$$

and, for every $Y \in T$

$$K_{Y,q} = \sum_{r \in R_\alpha(Y)} f_r(q) \qquad \text{and} \qquad w_{Y,q}(r) = \frac{f_r(q)}{K_{Y,q}}$$

where it is assumed that $K_{Y,q} \neq 0$.

3 MP Graphs and Biological Examples

The P metabolic algorithm was proven adequate in many cases of biological modeling we listed at the end of Section 1. Examples of biological models, formalized in terms of PM systems, will be collected in [5].

In an MP system two parts are clearly distinguishable: the *signature* and the *quantities*. The first part indicates the kinds of objects, the reactions and their regulation structure. The second part specifies the quantitative aspects which give meaning to the numbers which describe the evolution of systems. We represent the signature of metabolic P systems, in a way directly readable in

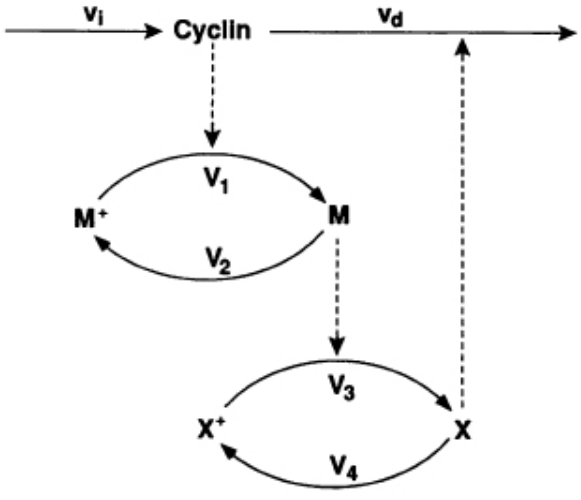


Fig. 1. The model provided by A. Goldbeter, from [17]

terms of PM algorithm, by means of graphs. Similar graphical formalisms were developed in the context of complex reaction networks (SNA, *Stoichiometric Network Analysis*, and MCA, *Metabolic Control Analysis* [10,11,37]). Formally, an *MP graph* is a structure $G = (T, R, F, E, C)$, where:

- T is the set of nodes representing types (we can think of each $t \in T$ as a container holding a certain amount of a peculiar kind of substance). Usually, we represent such kind of nodes as big circles labeled with the type of objects contained in it.
- R is the set of nodes representing biochemical reactions between types. We represent each of the nodes in R as a full bullet and we label it with the name of the reaction represented by that node.
- F is the set of nodes labeled by reaction maps. We represent this kind of nodes with full rectangles. These nodes are connected with a, possibly empty, set of circles (types) but they are also connected with exactly one bullet node.
- E is a set of nodes presenting input or output gates. It contains two different kind of nodes: *input gates* and *output gates*. Both of them have the triangular shape, where input gates have an arc exiting from a triangle vertex, and output gates have an arc entering in a triangle edge.
- C is a set of arcs between nodes. Edges are of two different kinds: plain edges or dashed edges.
  i) *Plain edges* connect types to biochemical reactions, in particular they specify *reactants* and *products* of the reaction. Arcs connecting reactants to reactions are depicted as lines while arcs connecting reactions to products appear as arrows (oriented arcs).

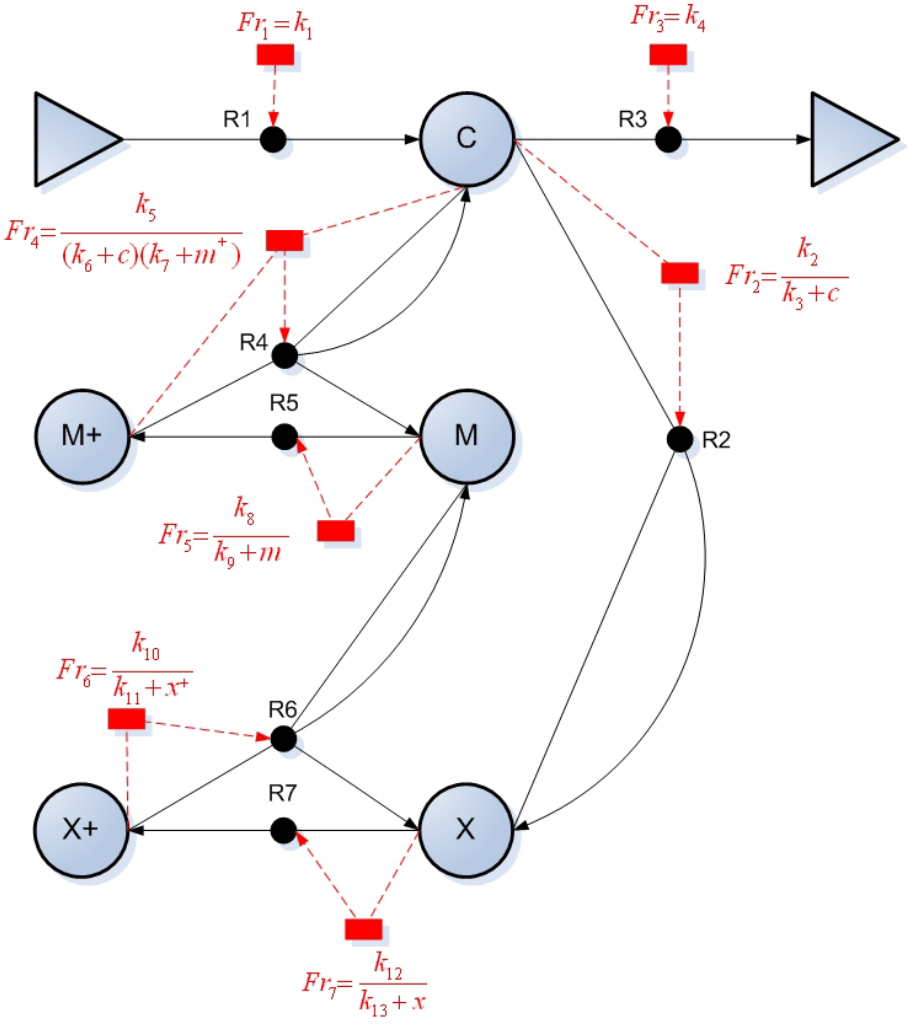


Fig. 2. A model of the mitotic oscillator of Figure 1 represented by a MP graph (from [28])

ii) *Edges* which connect types with square nodes (reactivity nodes) are depicted as dashed lines, while edges which connect square nodes with bullets are depicted as dashed arrows (see Figure 2).

The components E, C of an MP graph can be deduced from R and F, therefore we can omit them when we specify a graph.

Figure 2 shows an MP graph related to the mitotic oscillator in amphibian embryos, which is an important case study reported in [17]. Mitotic oscillations are a mechanism exploited by nature to regulate the onset of mitosis, that is, the process of cell division aimed at producing two identical daughter cells from

a single parent cell. More precisely, mitotic oscillations concern the fluctuation in the activation state of a protein produced by *cdc2* gene in fission yeasts or by homologous genes in other eukaryotes. The model considered here focuses on the simplest form of this mechanism, as it is found in early amphibian embryos. Here (see Figure 1) Cyclin is synthesized at a constant rate and triggers the transformation of inactive (M^+) into active (M) cdc2 kinase, by enhancing the rate of a phoshatase E_1. Another kinase reverts this modification. On the other hand, a kinase E_3 elicits the transformation from the inactive (X^+) to the active

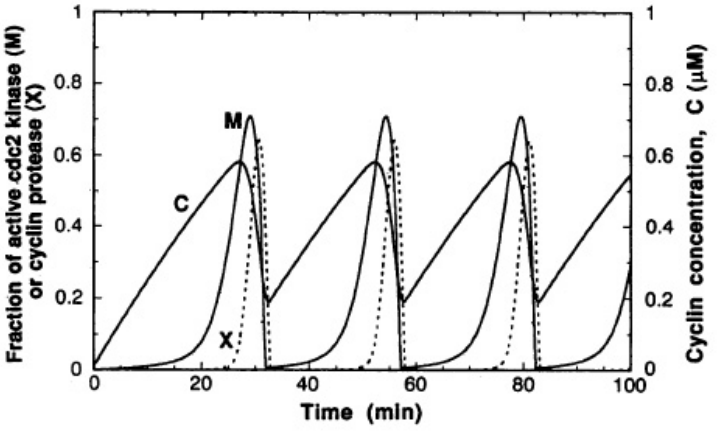


Fig. 3. A numerical solution of the set of differential equations (1) implementing the model provided by A. Goldbeter, from [17]

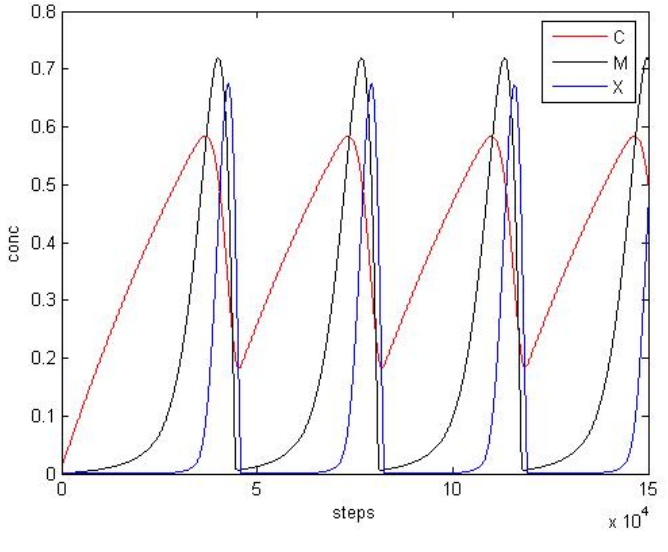


Fig. 4. The mitotic oscillator of Figure 1 computed means of an MP system evolution

(X) form of a protease that degrades cyclin, and this activation is reverted by a phoshatase E_4 (E_1, E_2, E_3, E_4 are not indicated in the figure, $v_i, v_d, V_1, V_2, V_3, V_4$ denote rates of the processes). The activation of *cdc2* kinase provides the formation of a complex known as M-phase promoting factor (or *MPF*). The complex triggers mitosis and the degradation of cyclin leads to the inactivation of the cdc2 kinase that brings the cell back to the initial conditions in which a new division cycle can take place. In yeasts and in somatic cells the cell cycle is subject to the control of many checkpoints, but the mechanism based on the activation-inactivation of cdc2 kinase remains the same [1]. The following equations are the differential model of its dynamics, where c, m, x are the percentages of C, M, X respectively ($1-m, 1-x$ are the percentages of M^+, X^+ respectively):

$$\begin{aligned} \frac{dc}{dt} &= v_i - v_d x \frac{c}{K_d + c} - K_d c \\ \frac{dm}{dt} &= V_1 \frac{(1-m)}{K_1 + (1-m)} - V_2 \frac{m}{K_2 + m} \\ \frac{dx}{dt} &= V_3 \frac{(1-x)}{K_3 + (1-x)} - V_4 \frac{x}{K_4 + x} \end{aligned} \tag{1}$$

Figure 3 gives a solutions of these equations obtained by numerical integration for some value of parameters given in [17]. The MP graph of Figure 2 was deduced from Goldbeter's model by means of a procedure given in [14].

Circadian rhythms are biochemical cycles evoked by variations in the expression level of genes. Such variations give rise to a surprisingly robust biological clock, synchronized with daylight and performing a complete cycle about every 24 hours. In the model of Drosophila melanogaster, circadian rhythms involve the oscillation of the Period (PER) and Timeless (TIM) proteins. According to

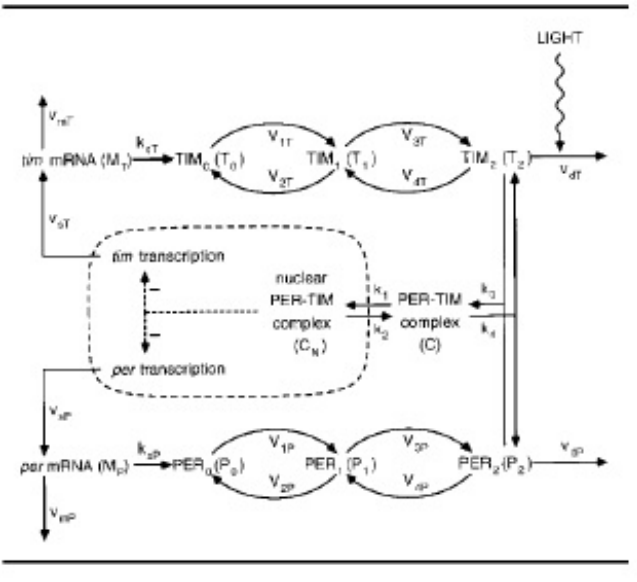


Fig. 5. Circadian rhythms in Drosophila, from [21]

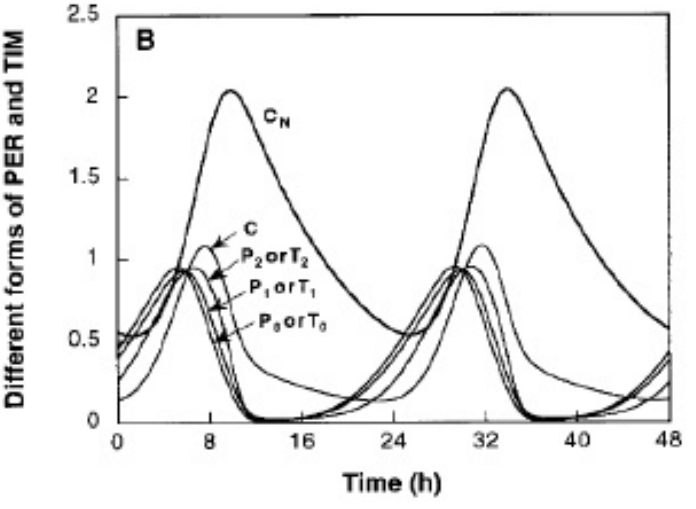


Fig. 6. Circadian rhythms in Drosophila: a numerical solution from [21]

this model the genes coding for PER and TIM proteins are inhibited by the presence of a PER-TIM protein complex. This complex is constituted by the two proteins, in the cytosol, under certain conditions, then it migrates inside the nucleus where becomes a PER and TIM suppressor. Gene expression and suppression result in a negative feedback network of signal transduction that has been formalized by a non-trivial system of nonlinear differential equations, devised by J. Leloup and A. Goldbeter [21,18]. A graphical scheme of the model is depicted in Figure 5 and a solution of differential model, for suitable values, is given in Figure 6. Also in this case we obtained similar solutions by using a suitable MP system deduced from the differential model of [18].

A general relationship between MP graphs and ODE holds. In fact, MP graphs transform naturally into ODE systems according to the mass principle, on which differential models are based on. The amount of a product generated by a reaction is proportional to the product of quantities of substrates (considered with their multiplicity). This idea is formalized by the following definition where the MP Notation 2 is assumed, x is the real variable $q(X)$, and x' denotes the derivative of variable x with respect to time.

Definition 4 (MP-ODE Transformation). *Let $G = (T, R, F)$ be an MP graph. For every $X \in T$, let x be the real variable associated to X. Then the following is the ODE-transform of G:*

$$x' = \sum_{r \in R} \{g_r(X) - h_r(X)\} f_r(q) \Pi(\alpha_r).$$

The following classes of MP systems play an important role in the relationship between ODE and MP systems.

Definition 3 (Non-cooperative MP System). *A non-cooperative MP system is an MP system whose rules are non-cooperative, i.e., $\alpha_r \in T$ for every rule r of the system.*

Definition 4 (Uniformly Inertial MP System). *For some $\phi \in \mathbb{R}$, an MP system is ϕ-uniformly inertial if the reaction map of any inertial rule of the system has the same constant value ϕ in any possible state.*

The following results can be proved as generalizations of those proved in [14].

Proposition 5. *Given an ODE, we can find (in many possible manners) an MP graph having the given ODE as its ODE-transform.*

Theorem 6. *The computation of a non-cooperative ϕ-uniformly inertial MP system converges, as $\phi \to \infty$, to the solution provided by the ODE system obtained by using MP-ODE transformation.*

Theorem 7. *For any MP system M, there exists a* non-cooperative *MP system M' having the same ODE-transform as M.*

Corollary 8. *Approximate solutions of autonomous ODE which describe metabolic systems can be found by computing the evolution of suitable MP systems.*

Figure 4 shows an MP solution, obtained by using the a non-cooperative system having as ODE-transform just the ODE 1. The similarity with Golbeter's solution 3 is really impressive and confirms the validity of the previous theorems, in a very significant biological model.

4 Conclusions

In many cases we were able to translate classical differential models into MP systems which provided similar results. Moreover, we showed that, under suitable hypotheses, this translation can be done in a systematic way [14], based on general relations on the two principles underlying these dynamical models: the differential ones assuming a time partition strategy and the MP ones assuming a mass partition strategy.

Evolutions of MP systems are discrete dynamics where important dynamical concepts could be investigated in the specific perspective of biomolecular dynamics. In fact, the approach developed in [26] could suggest useful criteria in the classification of dynamical features of biological relevance.

MP systems have several computational advantages with respect to the differential models, but their most important feature is their direct biological meaning and their structure where the reaction level and the regulation level are clearly interconnected but separated.

From the three principles underlying MPA and from Definition 3, it follows that the reactivities of rules in any state are univocally related to their reaction units and therefore to the variations of substances (from a state to the next

one). It would be interesting to find procedures that, under suitable hypotheses, could be able to recover from the knowledge of such variations the reaction units and then the reactivities, and finally, from the reactivities in different states, the reaction maps of rules.

The search for MP systems where these procedures can be defined, and computationally treated, is the main problem to solve for a systematic application of MP systems to complex dynamics. Without this possibility the construction of models is a very difficult task, which can be developed only with specific strategies depending on the particular cases.

An important aspect for future developments of our approach is the possibility to build directly a model from the data coming from the observation of biological phenomena. If we show that this task can be done in a systematic and efficient way, then MP systems will give a really useful instrument in the modeling of biological systems. The future development of theory and applications of these systems will tell us whether, or to which extent, they will satisfy this expectation.

References

1. B. Alberts and M. Raff. *Essential Cell Biology. An Introduction to the Molecular Biology of the Cell.* Garland Science, New York, 1997.
2. F. Bernardini and M. Gheorghe. Cell communication in tissue P systems: universality results. *Soft Computing*, 9(9):640–649, 2005.
3. F. Bernardini and V. Manca. P systems with boundary rules. In *Proc. 3rd Workshop on Membrane Computing*, *LNCS 2597*, 107–118, 2002. Springer.
4. D. Besozzi and G. Ciobanu. A P system description of the sodium-potassium pump. In G. Mauri, G. Păun, M.J. Pérez-Jiménez, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 5th International Workshop, WMC 2004*, *LNCS 3365*, 210–223, Springer, 2005.
5. L. Bianco. *Membrane Models of Biological Systems,* PhD Thesis, University of Verona, in preparation.
6. L. Bianco, F. Fontana, G. Franco, and V. Manca. P systems for biological dynamics. In [9], 81–126. 2006.
7. L. Bianco, F. Fontana, and V. Manca. Reaction-driven membrane systems. In L. Wang, K. Chen, and Y.-S. Ong, editors, *Advances in Natural Computation, First International Conference, ICNC 2005, Changsha, China, August 27-29, 2005, Proceedings, Part II*, *LNCS 3611*, 1155–1158. Springer, 2005.
8. L. Bianco, F. Fontana, and V. Manca. P systems with reaction maps. *International Journal of Foundations of Computer Science*, 17(1):27–48, 2006.
9. G. Ciobanu, G. Pău, and M. J. Pérez-Jiménez, editors. *Applications of Membrane Computing.* Springer, Berlin, Germany, 2006.
10. B.L. Clark. Stability of complex reaction networks. *Adv. Chem. Phys.*, 43, 1-216, 1983.
11. D.A. Fell. Metabolic control analysis: a survey of its theoretical and experimental development. *Biochemistry J.*, 286:313–330, 1992.
12. F. Fontana, L. Bianco, and V. Manca. P systems and the modeling of biochemical oscillations. In R. Freund, G. Păun, G. Rozenberg, and A. Salomaa, editors, *6th Workshop on Membrane Computing (WMC6)*, *LNCS 3850*, 199–208, Springer, 2005.

13. F. Fontana and V. Manca. Predator-prey dynamics in P systems ruled by metabolic algorithm. Submitted.
14. F. Fontana and V. Manca. Discrete solutions of differential equations by metabolic P systems. *Theoretical Computer Science*, to appear.
15. G. Franco and V. Manca. A membrane system for the leukocyte selective recruitment. In C. Martín-Vide, G. Mauri, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Proc. Int. Workshop, WMC2003, LNCS 2933*, 181–190, Springer, 2004.
16. R. Freund. Energy-controlled P systems. In G. Păun, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *Proc. Int. Workshop WMC-CdeA 2002*, *LNCS 2597*, 247–260, Springer, 2003.
17. A Goldbeter. A minimal cascade model for the mitotic oscillator involving cyclin and cdc2 kinase. *PNAS*, 88(20):9107–9111, 1991.
18. A Goldbeter. Computational approaches to cellular rhythms. *Nature*, 420:238-245, 2002.
19. A. Goldbeter. *Biochemical Oscillations and Cellular Rhythms.* Cambridge University Press, New York, 2004.
20. H. Kitano. Computational systems biology. *Nature*, 420:206–210, November 2002.
21. J.C. Leloup and A. Goldbeter. A model for circadian rhythms in Drosophila incorporating the formation of a complex between the PER and TIM proteins. *Journal of Biological Rhythms*, 13:70-87, 1998.
22. M. Maeda, S. Lu, G. Shaulsky, Y. Miyazaki, H. Kuwayama, Y. Tanaka, A. Kuspa, W. Loomis, Periodic signaling controlled by an oscillatory circuit that includes krotein Kinases ERK2 and PK. *Science*, 304, 875-304, 2004.
23. V. Manca. Rewriting and metabolism: A logical perspective. In G. Păun, editor, *Computing with Bio-Molecules*, Springer, 1998.
24. V. Manca and D.M. Martino. From string rewriting to logical metabolic systems. In G. Păun and A. Salomaa, editors, *Grammatical Models of Multi-Agent Systems*, Gordon and Breach Science Publishers, 1999.
25. V. Manca, L. Bianco, and F. Fontana. Evolutions and oscillations of P systems: Applications to biological phenomena. In G. Mauri, G. Păun, M.J. Pérez-Jiménez, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 5th International Workshop, WMC 2004*, *LNCS 3365*, 63–84, Springer, 2005.
26. V. Manca, G. Franco, and G. Scollo. State transition dynamics: basic concepts and molecular computing perspectives. In M. Gheorghe, editor, *Molecular Computational Models: Unconventional Approachers*, Chapter 2., 32-55, Idea Group Inc. UK, 2005.
27. V. Manca. Topics and problems in metabolic P systems. In G. Păun and M.J. Pérez-Jiménez, editors, *Proc. of the Fourth Brainstorming Week on Membrane Computing (BWMC4)*, Sevilla, Spain, Fenix Editora, 2006.
28. V. Manca, L. Bianco. Biological networks in metabolic P systems. Submitted.
29. C. Martin-Vide, G. Păun, and G. Rozenberg. Membrane systems with carriers. *Theoretical Computer Science*, 270:779–796, 2002.
30. A. Păun and G. Păun. The power of communication: P systems with symport/antiport. *New Generation Computing*, 20(3):295–306, 2002.
31. G. Păun. Computing with membranes. *J. Comput. System Sci.*, 61(1):108–143, 2000.
32. Gh. Păun. *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
33. G. Păun, Y. Suzuki, and H. Tanaka. P systems with energy accounting. *Int. J. Computer Math.*, 78(3):343–364, 2001.

34. Y. Suzuki, Y. Fujiwara, H. Tanaka, and J. Takabayashi. Artificial life applications of a class of P systems: Abstract rewriting systems on multisets. In C.S. Calude, G. Păun, G. Rozenberg, A. Salomaa editors, *Multiset Processing, Mathematical, Computer Science, and Molecular Computing Points of View*, LNCS 2235, 299–346. Springer-Verlag, Berlin, 2001.
35. Y. Suzuki and H. Tanaka. A symbolic chemical system based on an abstract rewriting system and its behavior pattern. *J. of Artificial Life and Robotics*, 6:129–132, 2002.
36. Y. Suzuki and H. Tanaka. Modelling p53 signaling pathways by using multiset processing. In G. Ciobanu, M.J. Pérez-Jiménez, and G. Păun, editors, *Applications of Membrane Computing*, 203–214. Springer, Berlin, 2006.
37. L.A. Segel and I.R. Cohen, editors. *Design Principles for the Immune System and Other Distributed Autonomous Systems*. Oxford University Press, 2000.
38. E.O. Voit. *Computational Analysis of Biochemical Systems*. Cambridge University Press, 2000.
39. The P Systems Web Page. `http://psystems.disco.unimib.it`

Modeling Signal Transduction Using P Systems

Andrei Păun[1], Mario J. Pérez-Jiménez[2], and Francisco J. Romero-Campero[2]

[1] Department of Computer Science/IfM, Louisiana Tech University
P.O. Box 10348, Ruston, LA 71272
apaun@latech.edu
[2] Department of Computer Science and Artificial Intelligence, University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
{marper, fran}@us.es

Abstract. Cellular signalling pathways are fundamental to the control and regulation of cell behavior. Understanding of biosignalling network functions is crucial to the study of different diseases and to the design of effective therapies. In this paper we present P systems as a feasible computational modeling tool for cellular signalling pathways that takes into consideration the discrete character of the components of the system and the key role played by membranes in their functioning. We illustrate these cellular models simulating the epidermal growth factor receptor (EGFR) signalling cascade and the FAS–induced apoptosis using a deterministic strategy for the evolution of P systems.

1 Introduction

The complexity of biomolecular cell systems is currently the focus of intensive experimental research, nevertheless the enormous amount of data about the function, activity, and interactions of such systems makes necessary the development of models able to provide a better understanding of the dynamics and properties of the systems.

A model is an abstraction of the real-world onto a mathematical/computational domain that highlights some key features while ignoring others that are assumed to be not relevant. A good model should have four properties: relevance, computability, understandability, and extensibility, [22]. A model must be relevant capturing the essential properties of the phenomenon investigated, and computable so it can allow the simulation of its dynamic behavior, as well as the qualitative and quantitative reasoning about its properties. An understandable model will correspond well to the informal concepts and ideas of molecular biology. Finally, a good model should be extensible to higher levels of organizations, like tissues, organs, organisms, etc., in which molecular systems play a key role.

P systems are an unconventional model of computation inspired by the structure and functioning of living cells which takes into consideration the discrete character of the quantity of components of the system by using rewriting rules on multisets of objects, that represent chemical substances, and strings, that represent the organization of genes on the genome. The inherent randomness in

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 100–122, 2006.

biological phenomena is captured by using stochastic strategies, [20]. We believe that P systems satisfy the above properties required for a good model.

Cellular signalling pathways are fundamental to the control and regulation of cell behavior. Understanding of biosignalling network functions is crucial to the study of different diseases and to the design of effective therapies. The characterization of properties about whole–cell functions requires mathematical/computational models that quantitatively describe the relationship between different cellular components.

Ordinary differential equations (ODEs) have been successfully used to model kinetics of conventional *macroscopic* chemical reactions. The approach followed by ODEs is referred as macroscopic chemistry since they model the average evolution of the concentration of chemical substances across the whole system. In this approach the change of chemical concentration over time is described for each chemical specie, implicitly assuming that the fluctuation around the average value of concentration is small relatively to the concentration. This assumption of homogeneity may be reasonable in some circumstances but not in many cases due to the internal structure and low numbers and non–uniform distributions of certain key molecules in the cell. While differential equations models may produce useful results under certain conditions, they provide a rather incomplete view of what is actually happening in the cell [2].

Due to the complexity of cellular signalling pathways, large number of linked ODEs are often necessary for a reaction kinetics model and the many interdependent differential equations can be very sensitive to their initial conditions and constants. Time delays and spatial effects (that play an important role in pathway behavior) are difficult to include in an ODE model [9], which are also very difficult to change and extend, because changes of network topology may require substantial changes in most of the basic equations [3].

Recently, different agent–based approaches are being used to model a wide variety of biological systems ([10], [12], [26]) and biological processes, including biochemical pathways [9].

The *microscopic* approach considers the molecular dynamics for each single molecule involved in the system taking into account their positions, momenta of atoms, etc. This approach is computationally intractable because of the number of atoms involved, the time scale and the uncertainty of initial conditions.

Our approach is referred as *mesoscopic* chemistry [25]. Like in the microscopic approach one considers individual molecules like proteins, DNA and mRNA, but ignores many molecules such as water and non-regulated parts of the cellular machinery. Besides, the position and momenta of the molecules are not modeled, instead one deals with the statistics of which reactions occur and how often. This approach is more tractable than microscopic chemistry but it provides a finer and better understanding than the macroscopic chemistry.

This paper is organized as follows. In the next section we present P systems as a framework for the specification of models of biosignalling cascades. A deterministic strategy for the evolution of P systems is described in Section 3. In Sections 4 and 5 a study of epidermal growth factor receptor (EGFR) signalling

cascade and of FAS–induced apoptotic signalling pathway are given. Finally, conclusions are presented in the last section.

2 P Systems: A Framework to Specify Biosignalling Cascades

In this paper we work with a variant of P systems of the form

$$\Pi = (O, L, \mu, M_1, M_2, \ldots, M_n, R_1, \ldots, R_n),$$

where:

- O is a finite alphabet of symbols representing objects (proteins and complexes of proteins);
- L is a finite alphabet of symbols representing labels for the compartments (membranes);
- μ is a membrane structure containing $n \geq 1$ membranes labeled with elements from L;
- $M_i = (w_i, l_i)$, $1 \leq i \leq n$, are pairs which represent the initial configuration of membrane i: $l_i \in L$ is its label, and $w_i \in O^*$ is the initial multiset.
- R_i, $1 \leq i \leq n$, are finite sets of rules associated with the membrane i which are of the form $u\,[\,v\,]_{l_i} \rightarrow u'\,[\,v'\,]_{l_i}$, where $u, v, u', v' \in O^*$ are finite multisets of objects and l_i is the label of membrane i.

Next, we discuss in more detail the rules that we will use in this paper, to model protein–protein interactions taking place in the compartmentalized structure of the living cell.

(a) *Transformation, complex formation and dissociation rules:*

$$\left.\begin{array}{l} [\,a\,]_l \rightarrow [\,b\,]_l \\ [\,a,\,b\,]_l \rightarrow [\,c\,]_l \\ [\,a\,]_l \rightarrow [\,b,\,c\,]_l \end{array}\right\} \quad \text{where } a, b, c \in O, \text{ and } l \in L$$

These rules are used to specify chemical reactions taking place inside a compartment of type $l \in L$; more specifically, they represent the transformation of a into b, the formation of a complex c from the interaction of a and b, and the dissociation of a complex a into b and c respectively.

(b) *Diffusing in and out:*

$$\left.\begin{array}{l} [\,a\,]_l \rightarrow a\,[\;\;]_l \\ a\,[\;\;]_l \rightarrow [\,a\,]_l \end{array}\right\} \quad \text{where } a \in O, \text{ and } l \in L$$

We use these types of rules when chemical substances move or diffuse freely from one compartment to another one.

(c) *Binding and debinding rules:*

$$\left.\begin{array}{l} a\,[\,b\,]_l \rightarrow [\,c\,]_l \\ \\ [\,a\,]_l \rightarrow b\,[\,c\,]_l \end{array}\right\} \quad \text{where } a, b, c \in O, \text{ and } l \in L$$

Using rules of the first type we can specify reactions consisting in the binding of a ligand swimming in one compartment to a receptor placed on the membrane surface of another compartment. The reverse reaction, debinding of substance from a receptor, can be described as well using the second rule.

(d) *Recruitment and releasing rules:*

$$\left.\begin{array}{l} a\,[\,b\,]_l \rightarrow c\,[\ ]_l \\ \\ c\,[\ ]_l \rightarrow a\,[\,b\,]_l \end{array}\right\} \quad \text{where } a, b, c \in O, \text{ and } l \in L$$

With these rules we represent the interaction between two chemicals in different compartments whereby one of them is recruited from its compartment by a chemical on the other compartment, and then the new complex remains in the latter compartment. In a releasing rule a complex, c, located in one compartment can dissociate into a and b, with remaining a in the same compartment as c, and b being released into the other compartment.

3 P Systems Using Deterministic Waiting Times Algorithm

In biological systems with a large number of molecules deterministic approaches are valid since the interactions between them follows the $\sqrt{n}$ law of physics, which states that randomness or fluctuation level in a system are inversely proportional to the square root of the number of particles.

Next, we present an *exact* deterministic strategy providing a semantic to the P systems defined before, that we will refer to as *deterministic waiting times algorithm*. It is based on the fact that *in vivo* chemical reactions take place in parallel in an asynchronous manner, i.e., different chemical reactions proceed at different reaction rates and the same reaction may also have different reaction rates at different times depending on the concentrations of reactants in the region.

In the deterministic waiting time strategy, the time necessary for a reaction to take place, called waiting time, is calculated and the rule or rules (chemical reaction) with the shortest waiting time is/are applied, changing the number of molecules in the respective compartments. In each step when there is a change in the number of a molecules in a compartment, the waiting time for the reactions "using" the changed molecule species has to be recalculated in that compartment.

By an exact deterministic method we mean that infinitesimal intervals of time are not approximated by Δt as it is the case in ODEs–based model, but we will

associate a waiting time, computed in a deterministic way, to each reaction and will use it to determine the order in which the reactions take place.

In our models biochemical reactions are used to describe the molecular interactions, and reversible complex formation reactions are frequent. In what follows we discuss how to compute mesoscopic rate constants from the macroscopic ones used in differential equations.

Our rules model reactions of the form:

$$A + B \underset{k_d}{\overset{k_a}{\rightleftharpoons}} C$$

This reversible reaction converges to an equilibrium in which the number of chemical species A, B and C remains constant. The equilibrium constant, K_{eq}, expresses the quantities of reactants A and B compared to complexes C once the equilibrium is reached; that is,

$$K_{eq} = \frac{[C]}{[A] \cdot [B]} \tag{1}$$

K_{eq} can also be computed using the association k_a and dissociation k_d rate constants:

$$K_{eq} = \frac{k_a}{k_d} \tag{2}$$

The association constant k_a determines the speed of the association reaction. It measures the number of chemicals A and B that form complexes C per mol and second. For the case of regulatory proteins the association rate constant k_a can be determined experimentally. K_{eq} can also be determined experimentally using (1) and therefore k_d can be computed using (2).

Alternatively, what it can be determined experimentally is Gibbs free energy ΔG, a notion from thermodynamics which measures the effort necessary for decomplexation. Gibbs free energy is related to the equilibrium constant K_{eq} as follows:

$$K_{eq} = \exp(\frac{-\Delta G}{R \cdot T}) \tag{3}$$

where $R = 1.9872\, cal\, mol^{-1}\, Kelvin^{-1}$ is the universal gas constant and T is the absolute temperature at which the experiments are performed.

Therefore from (2) and (3) the dissociation constant can be determined.

The rate constants k_a and k_d we have dealt with up to now are macroscopic, they do not depend on the actual number of molecules, but on concentration. Gillespie's algorithm and thus our approach uses mesoscopic rate constants referring to the actual number of molecules and they are determined from their macroscopic counterparts as follows:

$$c_a = \frac{k_a}{A \cdot V}, \qquad c_d = k_d$$

where $A = 6.023 \cdot 10^{23}$ is Avogadro's number and V is the cell volume. Note that we assume the cell volume to be constant while ignoring cell growth.

Given a P system, in this strategy each rule r (representing a chemical reaction) in each membrane m has associated a *velocity*, v_r, obtained by multiplying the mesoscopic rate constant c_r by the multiplicities of the reactants according to the mass action law. Then we compute the waiting time for the first execution of the rule r as $\tau_r = \frac{1}{v_r}$ and return the triple (τ_r, r, m).

Next, we give a detailed description of the *deterministic waiting times algorithm* providing the semantic of our P systems–based model:

- **Initialization**
  - ⋆ set time of the simulation $t = 0$;
  - ⋆ for every rule r associated with a membrane m in μ compute the triple (τ_r, r, m) by using the procedure described before; construct a list containing all such triples;
  - ⋆ sort the list of triple (τ_r, r, m) according to τ_r (in an ascendent order);
- **Iteration**
  - ⋆ extract the first triple, (τ_r, r, m) from the list (if there are several rules with the minimum waiting times, then we select all these rules);
  - ⋆ set time of the simulation $t = t + \tau_r$;
  - ⋆ update the waiting time for the rest of the triples in the list by subtracting τ_r;
  - ⋆ apply the rule(s) r only once updating the multiplicities of objects in the membranes affected by the application of the rule;
  - ⋆ for each membrane m' affected by the application of the rule(s) r, recalculate the waiting times of the rules which are in m';
  - ⋆ for each such rule, compare the new waiting times with the existing ones, and keep the smallest one among the two;
  - ⋆ sort the list of the new triples according to the waiting time;
  - ⋆ iterate the process.
- **Termination**
  - ⋆ Repeat the process until the time of the simulation t reaches or exceeds a preset maximal time of simulation.

Note that in this algorithm every rule in each membrane has a waiting time computed in a deterministic way that is used to determine the order in which the rules are executed. It is also worth mentioning that in this method the time step varies across the evolution of the system and it is computed in each step depending on the current state of the system.

This strategy has been implemented using Scilab, a scientific software package for numerical computations providing a powerful open computing environment for engineering and scientific applications [31]. This tool is available from [32].

4 Modeling EGFR Signalling

The epidermal growth factor receptor (EGFR) is provably the best understood receptor system, and computational models have played an important role in its elucidation. It seems clear that cells process the information before passing it to

the nucleus. The many different control points in the EGFR signalling pathway make it an excellent system for investigating how cells process contextual information. Computational models can play a crucial role for understanding this process [29].

In this section we study the EGFR signalling cascade where the deterministic waiting times algorithm is suitable for describing its evolution.

The epidermal growth factor receptor (EGFR) was the first found to have tyrosine–kinase activity and has been used in pioneering studies of biological processes such as receptor–mediated endocytosis, oncogenesis, mitogen–activated–protein–kinase (MAPK) signalling pathways, etc. [29].

Binding of the epidermal growth factor (EGF) to the extracellular domain of EGFR induces receptor dimerization and autophosphorylation of intracellular domains. Then a multitude of proteins are recruited starting a complex signalling cascade and the receptor follows a process of internalization, ubiquitination, and degradation in endosomals.

In our model we consider two marginal pathways and two principal pathways starting from the phosphorylated receptor.

In the first marginal pathway phospholipase C-γ (PLC_γ) binds to the phospholyrated receptor, then it is phosphorylated (PLC^*_γ) and released into the cytoplasm where it can be translocated to the cell membrane or desphosphorylated. In the second marginal pathway the protein PI3K binds to the phospholyrated receptor, then it is phosphorylated ($PI3K^*$) and released into the cytoplasm where it regulates several proteins that we do not include in our model.

Both principal pathways lead to activation of Ras-GTP. The first pathway does not depend on the concentration of the Src homology and collagen domain protein (Shc). This pathway consists of a cycle where the proteins growth factor receptor-binding protein 2 (Grb2) and Son of Sevenless homolog protein (SOS) bind to the phosphorylated receptor. Later the complex Grb2-SOS is released in the cytoplasm where it dissociates into Grb2 and SOS.

In the other main pathway Shc plays a key role, it binds to the receptor and it is phosphorylated. Then either Shc^* is released in the cytoplasm or the proteins Grb2 and SOS binds to the receptor yielding a four protein complex (EGFR-EGF2*-Shc*-Grb2-SOS). Subsequently, this complex dissociates into the complexes Shc^*-Grb2-SOS, Shc^*-Grb2 and Grb2-SOS which in turn can also dissociate to produce the proteins Shc^*, Grb2 and SOS.

Finally, Ras-GTP is activated by these two pathways and in turn it stimulates the Mitogen Activated Protein (MAP) kinase cascade by phosphorylating the proteins Raf, MEK and ERK. Subsequently, phosphorylated ERK regulates several cellular proteins and nuclear transcription factors that we do not include in our model.

There exist *cross-talks* between different parts and cycles of the signalling cascade which suggests a strong robustness of the system.

In Figure 1 it is shown a detailed graphical representation of the signalling pathway that we model in this paper.

Next, we present a P system–based model of the biosignalling cascade described above.

Our model consists of more that 60 proteins and complexes of proteins and 160 chemical reactions. We will not give all the details of the model. A complete description of $\mathbf{\Pi}_{EGF}$ with some supplementary information is available from the web page [32]. In what follows we give an outline of our model.

Let us consider the P system

$$\mathbf{\Pi}_{EGF} = (O, \{e, s, c\}, \mu, (w_1, e), (w_2, s), (w_3, c), \mathcal{R}_e, \mathcal{R}_s, \mathcal{R}_c),$$

where:

• **Alphabet:** In the alphabet O we represent all the proteins and complexes of proteins that take part in the signalling cascade simulated. Some of the objects from the alphabet and the chemical compounds that they represent are listed below.

Object	**Protein or Complex**
EGF	Epidermal Growth Factor
EGFR	Epidermal Growth Factor Receptor
EGFR-EGF_2	Dimerisated Receptor
EGFR-EGF_2^*-Shc	EGFR-EGF_2^* and Shc complex
⋮	⋮
MEK	Mitogenic External Regulated Kinase
ERK	External Regulated Kinase

• **Membrane Structure:** In the EGFR signalling cascade there are three relevant regions, namely the *environment*, the *cell surface* and the *cytoplasm*. We represent them in the membrane structure as the membranes labeled with e for the environment, s for the cell surface, and c for the cytoplasm. The skin of the structure is the environment, the cell surface is the son of the environment and the father of the cytoplasm.

• **Initial Multisets:** In the initial multisets we represent the initial number of molecules of the chemical substances in the environment, the cell surface, and the cytoplasm. These estimations has been obtained from [13,24].

$$w_e = \{EGF^{20000}\}$$
$$w_s = \{EGFR^{25000}, Ras\text{-}GDP^{20000}\}$$
$$w_c = \{Shc^{25000}, PLC_\gamma^{15000}, PI3K^{5000}, SOS^{4000}, Grb2^{8000}, TP_1^{10000}, TP_2^{45000}, TP_3^{45000}, TP_4^{12500}, Raf^{8000}, MEK^{40000}, ERK^{40000}, P_1^{8000}, P_2^{8000}, P_3^{30000}\}$$

• **Rules:** Using rules we model the 160 chemical reactions which form the signalling cascade.

As it can be seen in the initial multisets specified before, in the system of the EGFR signalling cascade the number of molecules is quite large, hence as a consequence of the $\sqrt{n}$ law important fluctuations and stochastic behavior are

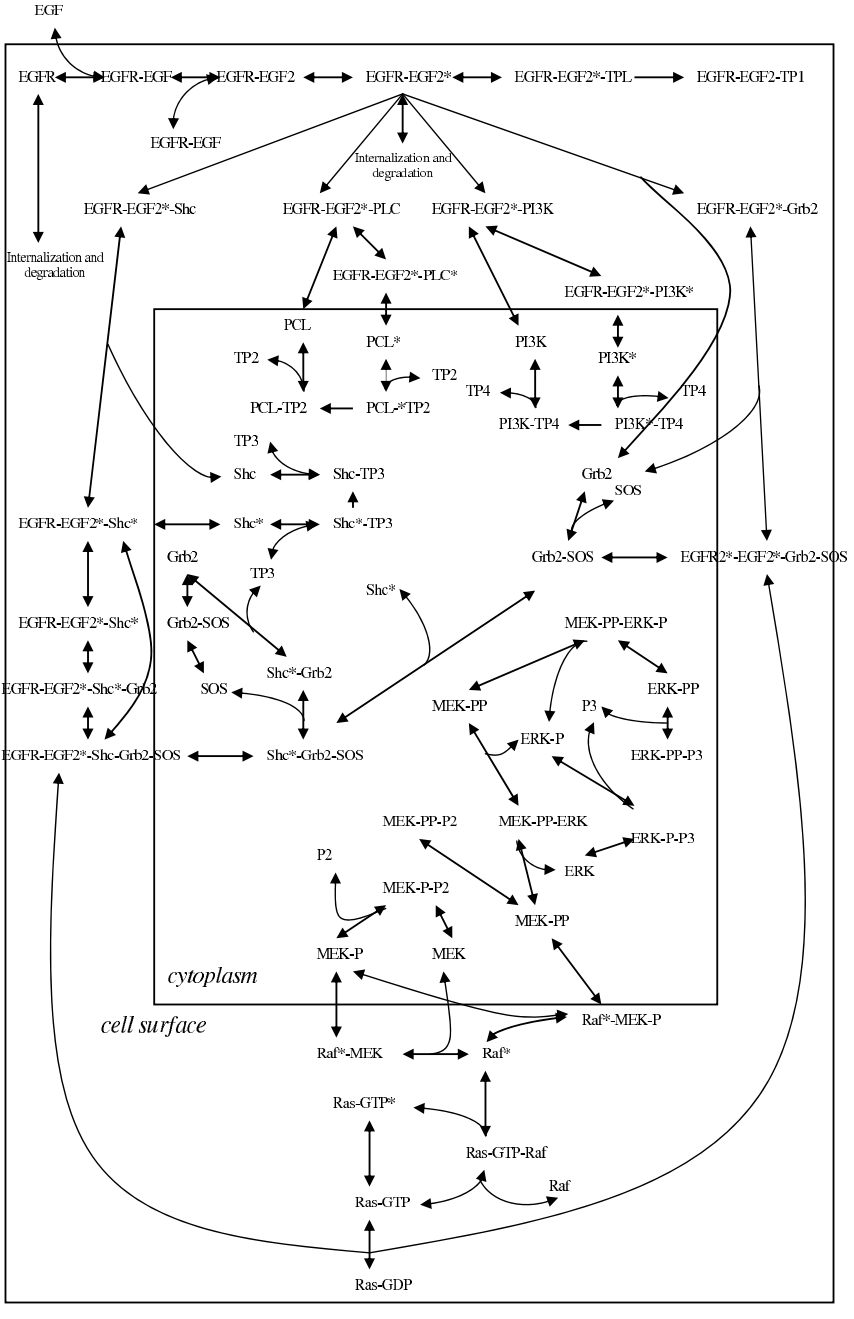


Fig. 1. EGFR Signalling Cascade

not expected in the evolution of the system. Because of this we have chosen the deterministic waiting times algorithm as the strategy for the evolution of the P system $\mathbf{\Pi}_{EGF}$.

Next, we show two examples of rules of the system.

The set of rules associated with the environment, $\mathcal{R}_e$, consists only of one rule r which models the binding of the signal, EGF, to the receptor $EGFR$.

$$EGF\,[\,EGFR\,]_s \;\rightarrow\; [\,EGF\text{-}EGFR\,]_s\;, c_r$$

The meaning of the previous rule is the following: the object EGF in the membrane containing the membrane with label s (the environment), and the object $EGFR$ inside the membrane with label s (the cell surface) are replaced with the object $EGFR$-EGF in the membrane with label s; this object represents the complex receptor-signal on the cell surface. We associate the mesoscopic rate constant c_r, which measures the affinity between the signal and the receptor.

The deterministic waiting times algorithm is used in the evolution of the system and the waiting time associated to this rule will be computed using the next formula:

$$\tau_r = \frac{1}{c_r \cdot |EGF| \cdot |EGFR|}$$

One example from the set of rules $\mathcal{R}_s$ associated with the cell surface is the rule r' concerning to the dimerisation of the receptor, that is the formation of a complex consisting of two receptors:

$$[\,EGFR,\, EGFR\,]_s \;\rightarrow\; [\,EGFR_2\,]_s\;, c_{r'}$$

When this rule r' is executed two objects $EGFR$ representing receptors are replaced with one object $EGFR_2$, representing a complex formed with two receptors, in the membrane with label s, the cell surface. The mesoscopic rate constant $c_{r'}$ is used to computed the waiting time:

$$\tau_{r'} = \frac{1}{c_{r'} \cdot |EGFR|^2}$$

4.1 Results and Discussions

Using Scilab we ran some experiments; in what follows we present some of the results obtained.

In Figure 2 it is depicted the evolution of the number of autophosphorylated receptors and in Figure 3 the number of doubly phosphorylated MEK (Mitogen External Kinase), one of the target proteins of the signalling cascade that regulates some nuclear transcription factors involved in the cell division.

Note that the activation of the receptor is very fast reaching its maximum within the first 5 seconds and then it decays fast to very low levels; on the other hand the number of doubly phosphorylated MEK is more sustained around 3 nM. These results agree well with empirical observations, see [13,24].

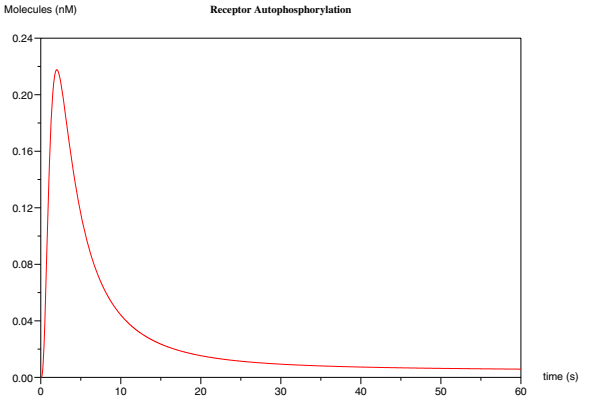


Fig. 2. Autophosphorylated EGFR evolution

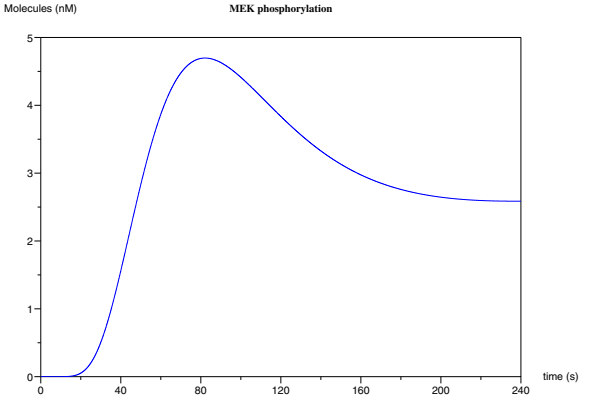


Fig. 3. Doubly phosphorylated MEK evolution

In tumors it has been reported an over expression of EGF signals in the environment and of EGFR receptors on the cell surface of cancerous cells. Here we investigate the effect of different EGF concentrations and number of receptors on the signalling cascade.

First, we study the effect on the evolution of the number of autophosphorylated receptors and doubly phosphorylated MEK of a range of signals, EGF, from 100 nM to 2000 nM.

In Figure 4, it can be seen that the receptor autophosphorylation is clearly concentration dependent showing different peaks for different number of signals in the environment. According to the variance in the receptor activation it is intuitive to expect different cell responses to different EGF concentrations. Here we will see that this is not the case.

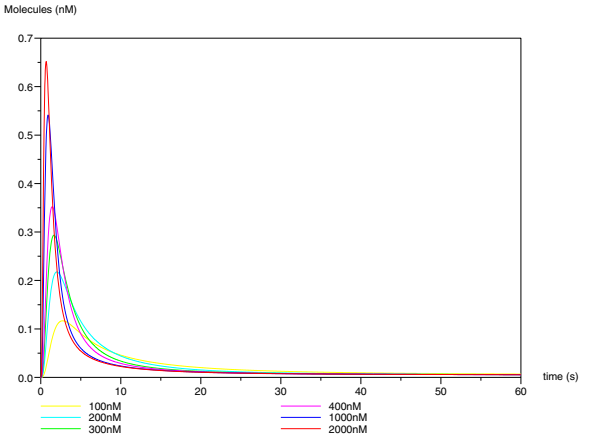


Fig. 4. Receptor autophosphorylation for different environmental EGF concentrations

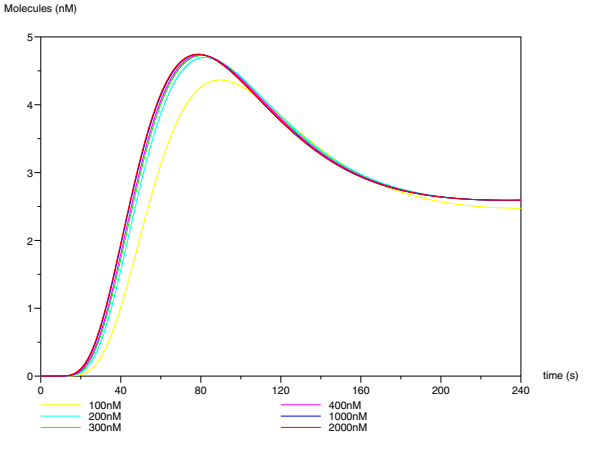


Fig. 5. MEK phosphorylation for different environmental EGF concentrations

From Figure 5 we can observe that the number of doubly phosphorylated MEK does not depend on the number of signals in the environment. That is, the perturbation of EGF level in the environment has no impact on MEK activation. So, the system is not sensitive to increases in EGF level in the environment (*Sensitivity analysis* is a mathematical technique term associated with the use of a computational model to predict the effects of the variation of a single component in the model).

This shows the surprising robustness of the signalling cascade with regard to the number of signals from outside due to EGF concentration. The signal is either attenuated or amplified to get the same concentration of one of the most relevant kinases in the signalling cascade, MEK. Note that after 100 seconds, when the

response gets sustained, the lines representing the response to different external EGF concentrations are identical.

Now we analyze the effect on the dynamics of the signalling cascade of different numbers of receptors on the cell surface.

In Figure 6, it is shown the evolution of the number of doubly phosphorylated MEK when there is 100 nM and 1000 nM of receptors on the cell surface. Note that now the response is considerably different; the number of activated MEK is greater when there is an over expression of receptors on the cell surface. As a consequence of this high number of activated MEK the cells will undergo an uncontrolled process of proliferation. Thus, Figure 6 shows the sensitivity of MEK activation to increases in EGFR level on the cell surface.

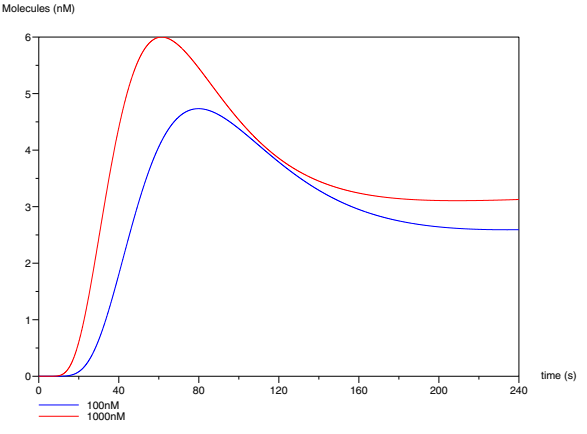


Fig. 6. MEK phosphorylation for different number of receptors

The key role played by the over expression of EGFR on the uncontrolled growth of tumors has been reported before, and, as a consequence of this, EGFR is one of the main biological targets for the development of novel and successful therapies against cancer and continue to be a source of discoveries about the cell signalling mechanisms involved in development, tissue homeostasis, and disease [28,14].

There are different strategies to inhibit the over expression of EGFR on the cell surface but the most developed ones are the *monoclonal antibodies* (that bind the external domain of the receptor competing against their natural ligands), and the *molecules with low molecular weight* (that inhibit the tyrosine–kinase activity of the receptor at the intracellular level).

Finally, we stress that for this system we have used a deterministic approach obtaining results that agree well with experimental data. This is not always the case, for instance in [20] a system is shown (the Quorum Sensing system in Vibrio Fischeri) where a stochastic approach is necessary to describe properly its behavior.

5 Modelling FAS–Apoptosis

There are basically two mechanisms of cell death, *necrosis* and *apoptosis*. Necrosis is a form of cell death that usually occurs when cells are damaged by injury. A disruption of the cell membrane is produced and intracellular materials are released. In contrast to necrosis, apoptosis is carried out in an ordered sequence of events that culminates in the suicide of the cell, and without releasing intracellular materials from the dying cells.

The term *apoptosis* (also known as *programmed cell death*) was coined by Kerr, Wyllie and Currie [8] as a means of distinguishing a morphologically distinctive form of cell death which was associated with normal physiology.

Apoptosis occurs during organ development, it plays an important role in cellular homeostasis [11], and it is a cellular response to a *cellular insult* that starts a cascade of apoptotic signals, both intracellular and extracellular, which converge on the activation of a group of apoptotic–specific proteases called caspases. The apoptotic mechanism include condensation of cell contents, DNA fragmentation into nucleosomal fragments, nuclear membrane breakdown, and the formation of apoptotic bodies that are small membrane–bound vesicles phagocytosed by neighboring cells [15]. Apoptosis protects the rest of the organism from a potentially harmful agent and disregulation of apoptosis can contribute to the development of autoimmune diseases and cancers. Apoptosis can also be induced by anticancer drugs, group factor deprivation, and irradiation.

The family of proteases that mediates apoptosis is divided into two subgroups. The first group consists of caspase 8, caspase 9, and caspase 10, and they function as initiators of the cell death process. The second group contains caspase 3, caspase 6, and caspase 7, and they work as effectors. The other effector molecule in apoptosis is Apaf-1, which, together with cytochrome c, stimulates the processing of pro-caspase 9 to the mature enzyme.

The other regulators of apoptosis are the Bcl2 family members, divided into three subgroups based on their structure. Members of the first subgroup, represented by Bcl2 and Bcl-xL, have an anti-apoptotic function. The second subgroup, represented by Bax and Bak, and the third subgroup, represented by Bid and Bad, are pro-apoptotic molecules.

Apoptotic death can be triggered by a wide variety of stimuli. Among the more studied death stimuli are DNA damage which in many cells leads to apoptotic death via a pathway dependent on p53, and the signalling pathways for FAS-induced apoptosis that was shown to be one of the most relevant processes for understanding and combating many forms of human diseases such as cancer, neurodegenerative diseases (Parkinson's disease, Alzheimer, etc.), AIDS and ischemic stroke.

Fas (also called CD95 or APO–1) is a cell surface receptor protein with an extracellular region, one transmembrane domain, and an intracellular region. Fas belongs to the tumor necrosis factor/nerve growth factor (TNT/NGF) cytokine receptor family. Activation of Fas through binding to its ligands, induces apoptosis in the Fas bearing cell. Fas induced–apoptosis starts from the Fas ligand

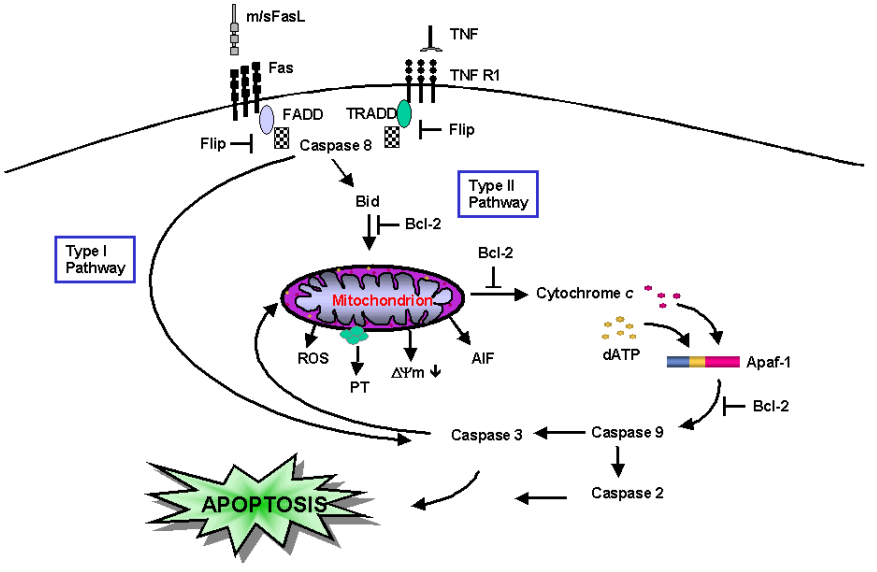


Fig. 7. FAS signalling pathways, from [1]

binding to Fas receptors and ends in the fragmentation of genomic DNA, which is used as a hallmark of apoptosis.

Fas ligands usually exist as trimers and bind and activate their receptors by inducing receptor trimerisation. This creates a clustering of Fas that is necessary for signalling. In its intracellular region, Fas contains a conserved sequence called a *death domain*. Activated receptors recruit adaptor molecules (such as FADD, Fas–associating protein with death domain) which interacts with the death domain on the Fas receptor and recruit procaspase 8 to the receptor complex, where it undergoes autocatalytic activation cleaving and releasing active caspase 8 molecules intracellularly. Activated caspase 8 can activate caspase 3 through two different pathways that have been identified by Scaffidi et al. [23], and are referred to as type I (*death receptor pathway*) and type II (*mitochondrial pathway*), where caspases play a crucial role for both the initiation and execution apoptosis.

The pathways diverge after activation of initiator caspases and converge at the end by activating executor caspases. In the type I pathway, initiator caspase (caspase 8) cleaves procaspase 3 directly and activates executor caspase (caspase 3).

In the type II pathway, a more complicated cascade is activated involving the disruption of mitochondrial membrane potential and it is mediated by Bcl2 family proteins that regulate the passage of small molecules which activate caspase cascades through the mitochondrial transition pore. More specifically (see Figure 8), caspase 8 cleaves Bid (Bcl2 interacting protein) and its COOH–terminal part translocates to mitochondria where it triggers cytochrome c release. The released cytochrome c bind to Apaf–1 (apoplectic protease activating factor)

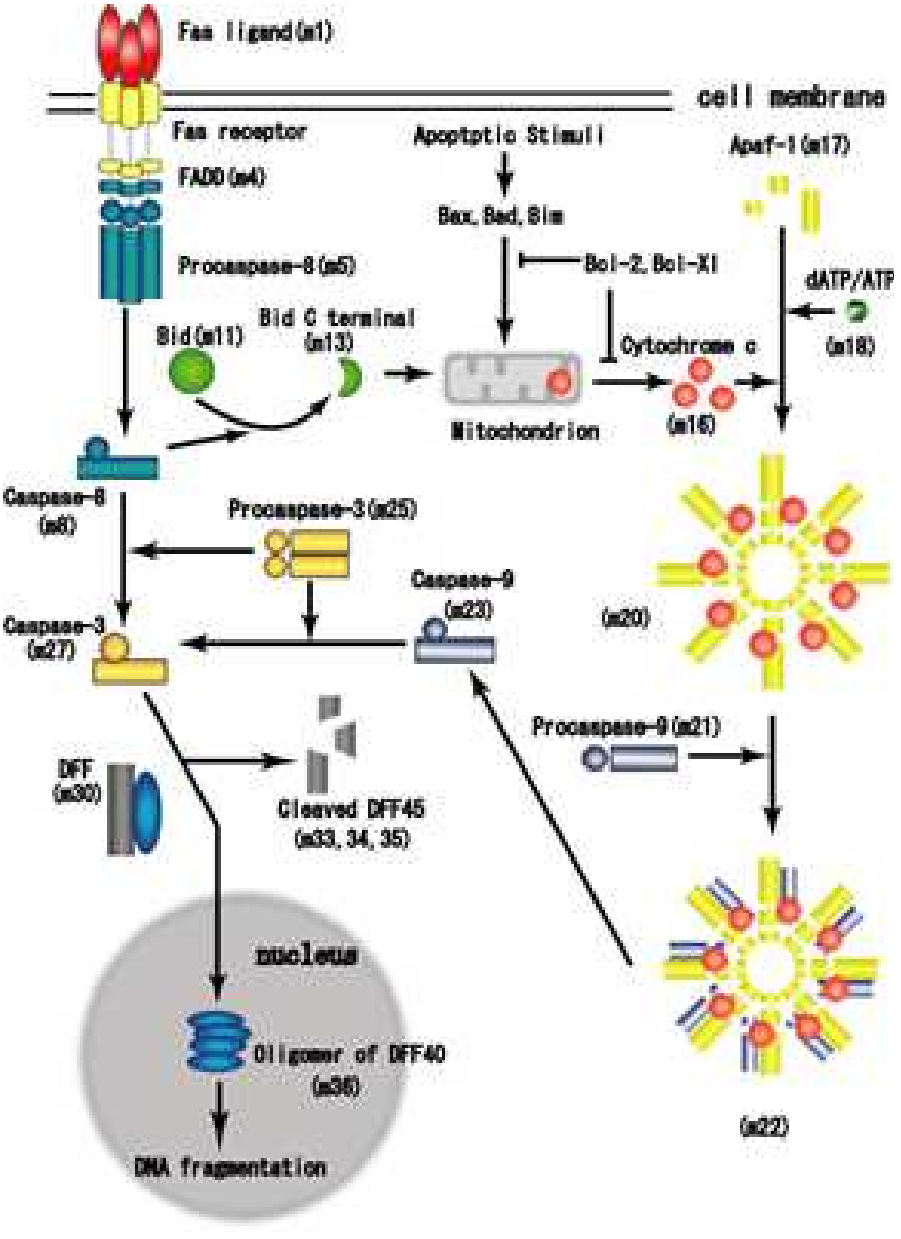


Fig. 8. Details of FAS signalling pathways, from [15]

togegther with dATP and procaspase 9 and activate caspase 9. The caspase 9 cleaves procaspase 3 and activates caspase 3.

The executor caspase 3 cleaves DFF (DNA fragmentation factor) in a heterodimeric factor of DFF40 and DFF45. Cleaved DFF45 dissociates from DFF40,

inducing oligomerisation of DFF40. The active DFF40 oligomer causes the internucleosomal DNA fragmentation.

Despite many molecular components of these apoptotic pathways have been identified, a better understanding of how they work together into a consistent network is necessary. A way to understand complex biological processes, in general, and the complex signalling behaviour of these pathways, in particular, is by modelling them in a computational framework and simulating them in electronic computers.

In [6] the two pathways activated by FAS starting with the stimulation of FASL (FAS ligand) until the activation of the effector caspase 3, have been modelled using ordinary differential equations in which biochemical reactions where used to describe molecular interactions.

In this section we present a P system using a deterministic waiting times algorithm for modelling FAS induced apoptosis, implementing all the rules described in [6] for both pathways.

Our model consists of 53 proteins and complexes of proteins and 99 chemical reactions. We will not give all the details of the model. A complete description of $\mathbf{\Pi}_{FAS}$ with some supplementary information can be found in the web page [32]. In what follows we give an outline of our model.

Let us consider the P system

$$\mathbf{\Pi}_{FAS} = (O, \{e, s, c, m\}, \mu, (w_1, e), (w_2, s), (w_3, c), (w_4, m), \mathcal{R}_e, \mathcal{R}_s, \mathcal{R}_c, \mathcal{R}_m)$$

where:

• **Alphabet:** In the alphabet O we represent all the proteins and complexes of proteins that take part in the signalling cascade simulated. Some of the objects from the alphabet and the chemical compounds that they represent are listed below.

Object	**Protein or Complex**
FAS	Fas protein
FASL	Fas Ligand
FADD	Fas–associating protein with death domain
⋮	⋮
Apaf	Apoptotic protease activating factor
Smac	Second mitochondria–derived activator of caspase
XIAP	X–linked inhibitor of apoptosis protein

• **Membrane Structure:** In the FAS signalling pathways there are four relevant regions, namely the *environment*, the *cell surface*, the *cytoplasm* and the *mitochondria*. We represent them in the membrane structure as the membranes labeled with: e for the environment, s for the cell surface, c for the cytoplasm, and m for the *mitochondria*. The skin of the structure is the environment, the cell surface is the son of the environment, the father of the cytoplasm, and the grandfather of the mitochondria.

• **Initial Multisets:** In the initial multisets we represent the initial number of molecules of the chemical substances in the environment, the cell surface, the cytoplasm, the mitochondria. These estimations has been obtained from [6].

$$\begin{aligned} w_e &= \{FASL^{12500}\} \\ w_s &= \{FAS^{6023}\} \\ w_c &= \{FADD^{10040}, CASP8^{20074}, FLIP^{48786}, CASP3^{120460}, Bid^{15057}, \\ &\quad Bax^{50189}, XIAP^{18069}, Apaf^{60230}, CASP9^{12046}\} \\ w_m &= \{Smac^{60230}, Cyto.c^{60230}, Bcl2^{45172}\} \end{aligned}$$

• **Rules:** Through the rules we model the 99 chemical reactions which form the signalling pathways. The rules can be found in [5] and they are described in our model as in the case of the system $\mathbf{\Pi}_{EGFR}$ (with different rules in the alternative cases of type II pathway in next subsection).

The set of rules associated with the environment, $\mathcal{R}_e$, consists only of one rule r_1 which models the binding of the FAS ligand to the receptor FAS.

$$FASL\ [\ FAS\ ]_s \rightarrow [\ FASC\ ]_s\ , c_{r_1}$$

The meaning of the previous rule is the following: the object $FASL$ in the membrane containing the membrane with label s (the environment), and the object FAS inside the membrane with label s (the cell surface) are replaced with the object $FASC$ in the membrane with label s; this object represents the complex receptor-signal on the cell surface. We associate the kinetic constant k_1, which measures the affinity between the signal and the receptor.

The deterministic waiting times algorithm is used in the evolution of the system and the waiting time associated to this rule will be computed using the next formula:

$$\tau_{r_1} = \frac{1}{c_{r_1} \cdot |EGF| \cdot |EGFR|}$$

5.1 Results and Discussions

We implemented in Java a preliminary *simulator* for the P system. It accepts as input an SBML (Systems Biology Markup Language) file containing the rules to be simulated and initial concentrations for the molecules in the system.

We compared our results with both the experimental data and with the ODEs simulation data reported in [6].

One of the major proteins in the pathway, caspase 3 was compared to the experimental data. In the ODEs simulation, caspase 3 was activated at 4 hours, and it was considered close to the experimental results where it was obtained that it activated at 6 hours (see Figure 9).

The same pathway is modeled in the membrane computing framework using the same reactions and initial conditions. The caspase 3 activation dynamics is studied when Bcl2 is at baseline value. Caspase 3 is activated in our simulator

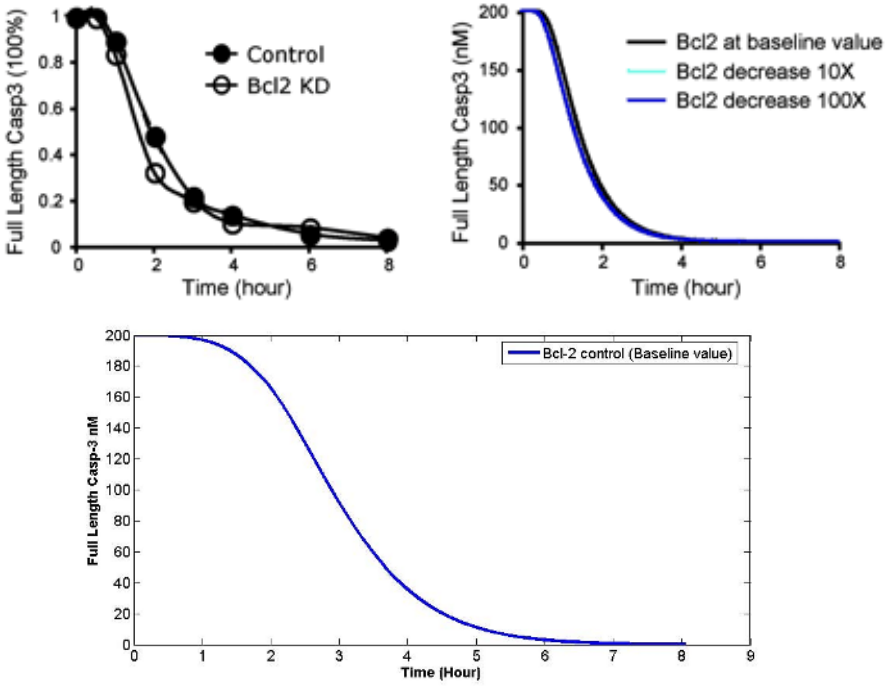


Fig. 9. Comparison between experimental data (top left, from [6]), previous ODE simulation data (top right, [6]) and the P system simulation data (down)

after about 7 hours which is a very good approximation of the experimental data and it improves the results obtained in the ODEs simulation [6].

There are cells (as thymocytes and fibroblasts) which are not sensitive to Bcl2 over expression as described in [23]. In these cells caspase 8 directly activates caspase 3.

Scaffidi et al. has suggested in [23] that the type of pathway activated by Fas is chosen based on the concentration of caspase 8 generated in active form following FASL binding. If the concentration of activated caspase 8 is high, then the caspase 3 is activated directly, on the other hand, if the concentration of activated caspase 8 is low, the type II pathway is chosen so that the system is amplifying the death signal through the mitochondria to be able to induce the cell death.

To check this hypothesis, the active caspase 8 formation is increased by having the initial concentration of caspase 8 set to a value 20 times greater than its baseline value while everything else was kept the same in the system. We performed the same simulation with the increase in caspase 8 initial concentration, and this resulted in faster caspase 3 activation also in our simulation; this agrees well with the results obtained in [23].

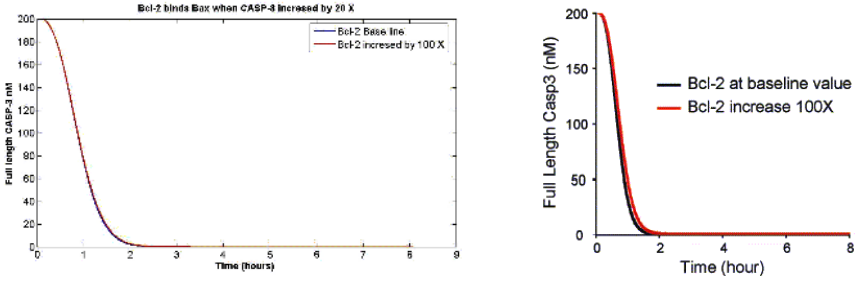


Fig. 10. Left – the P system simulation, right – the ODE simulation, from [6], for the change in caspase 8 initial concentration

The Bcl2 concentration is also increased 100 times to test the sensitivity of caspase 3 activation to Bcl2. Figure 10 shows that the caspase 3 activation is not sensitive to increases in Bcl2 concentration, when pathway of type I is chosen.

Bcl2 is known to block the mitochondrial pathway; however, it is not clear the mechanism through which Bcl2 can block the pathway of type II. Next, we analyze the caspase 3 activation kinetics in this type of pathway by considering different mechanisms to block the mitochondrial pathway suggested in [4], [16] and [27]: Bcl2 might bind with (a) Bax, (b) Bid, (c) tBid, or (d) bind to both Bax and tBid.

We design four different P systems having the rules:

- $r_1, \ldots, r_{95}, r_{96}, r_{97}$ for modeling the case (a).
- $r_1, \ldots, r_{95}, r'_{96}, r'_{97}$ for modeling the case (b).
- $r_1, \ldots, r_{95}, r''_{96}, r''_{97}$ for modeling the case (c).
- $r_1, \ldots, r_{97}, r_{98}, r_{99}$ for modeling the case (d).

All the other rules remain the same for all the cases (see [5] for details).

Let us note that this example shows the modularity of P systems–based model: small behavioral changes in the biosignalling cascade causes small changes in the designs of the P systems.

The dynamics of caspase 3 activation is studied by varying the Bcl2 concentration 10 times or 100 times the baseline value. It was concluded that Bcl2 binding to both Bax and tBid is the most efficient mechanism for the pathway in comparison with the results obtained for the cases (a), (b) or (c). The same conclusions were obtained also after using our simulator for all the previous changes in the pathway.

Figure 11 shows only the case (d) as a comparison between the ODE simulator and the P system simulator. It can seen the sensitivity of the caspase 3 activation to increases in Bcl2 level, when Bcl2 is able to bind to both Bax and tBid, and when mitochondrial pathway is selected.

Next table presents a summary about the sensitivity analysis of caspase 3 activation to over expression of Bcl2 in function of the pathway selected.

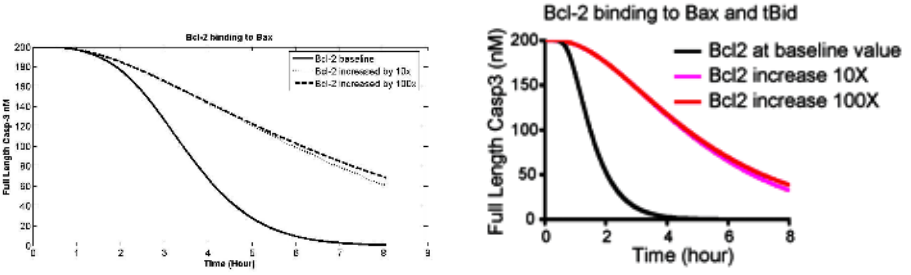


Fig. 11. Left – the P system simulation, right – the ODE simulation, from [6]

	Activation Caspase 3 (with over expression of Bcl2)
Type I (death receptor pathway)	*Insensitivity*
Type II (mitochondrial pathway)	*Sensitivity*

6 Conclusions

In this paper we have presented P systems as a new computational modeling tool to study the dynamic behavior of integrated signalling systems through a mesoscopic chemistry approach.

P systems are also a general specification of the biological phenomena that can be evolved using different strategies/algorithms. A *deterministic waiting times algorithm* has been introduced, and it is based on the fact that *in vivo* chemical reactions take place in parallel and in an asynchronous manner.

That strategy has been illustrated with the simulation of two relevant biological phenomena: the EGFR signalling cascade and the signalling pathways for FAS–induced apoptosis. In the line of [29] we think that the success of using P systems–based model for simulating biosignalling cascades can be a guide to combining models and experiments to understand complex biological processes as integrated systems.

Our results show a good correlation with the experimental data reported in the literature and with simulators based on ODEs. So, they support the reliability of P systems as computational modeling tools to produce postdiction, and perhaps they will be able to produce plausible predictions.

Acknowledgement

The author wishes to acknowledge the support of the project TIN2005-09345-C04-01 of the Ministerio de Educación y Ciencia of Spain, co–financed by FEDER funds, and of the project of Excellence TIC 581 of the Junta de Andalucía.

References

1. Alimonti, J.B., Ball, T.B., Fowke, K.R. Mechanisms of CD4+ T lymphocyte cell death in human immunodeficiency virus infection and AIDS. *Journal of General Virology*, **84** (2003), 1649–1661.
2. Bhalla, U.S., Iyengar, R., Emergent properties of networks of biological signaling pathways. *Science*, **283** (1999), 381–387.
3. Blossey, R., Cardelli, L., Phillips, A. A compositional approach to the stochastic dynamics of gene networks. *Transactions on Computational Systems Biology*, IV, Lecture Notes in Computer Science, **3939** (2006), 99–122.
4. Cheng, E.H., Wei, M.C., Weiler, S., Flavell, R.A., Mak, T.W., Lindsten, T., Korsmeyer, S.J. BCL-2, BCL-XL sequester BH3 domain-only molecules preventing BAX- and BAK-mediated mitochondrial apoptosis. *Molecular Cell*, **8** (2001), 705–711.
5. Cheruku, S., A. Păun, F.J. Romero, M.J. Pérez–Jiménez, O.H. Ibarra. Simulating FAS–induced apoptosis by using P systems. *Proceedings of the First International Conference on Bio–Inspired Computing: Theory and Applications*, Wuhan, China, September, 18–22, 2006.
6. Hua, F., Cornejo, M., Cardone, M., Stokes, C., Lauffenburger, D. Effects of Bcl-2 levels on FAS signaling-induced caspase-3 activation: Molecular genetic tests of computational model predictions. *The Journal of Immunology*, **175**, 2 (2005), 985–995 and correction **175**, 9 (2005), 6235–6237.
7. Ibarra, O.H., Păun, A. Counting time in computing with cells. *Proceedings of DNA Based Computing, DNA11*, London, Ontario, 25–36, 2005.
8. Kerr, J.F., Wyllie, A.H., Currie, A.R. Apoptosis: a basic biological phenomenon with wide-ranging implications in tissue kinetics. *British Journal Cancer*, **26** (1972), 239.
9. Pogson, M., Smallwood, R., Qwarnstrom, E., Holcombe, Formal agent–based of intracellular chemical interactions. *BioSystems*, **85**, 1 (2006), 37–45.
10. Holcombe, M., Gheorghe, M., Talbot, N. A hybrid machine model of rice blast fungus, Magnaphorte Grisea. *BioSystems*, **68**, 2–3 (2003), 223–228.
11. Jaatela, M. Multiple cell death pathways as regulators of tumour initiation and progression. *Oncogene*, **23** (2004), 2746–2756.
12. Jackson, D., Holcombe, M., Ratnieks, F. Trail geometry gives polarity to ant foraging networks. *Nature* **432** (2004), 907–909.
13. Moehren G., Markevich, N., Demin, O., Kiyatkin, A., Goryanin, I., Hoek, J.B., Kholodenko, B.N. Temperature dependence of the epidermal growth factor receptor signaling network can be accounted for by a kinetic model, *Biochemistry* **41** (2002), 306–320.
14. Moghal, N., Sternberg, P.W. Multiple positive and negative regulators of signaling by the EGFR receptor. *Curr. Opin. Cell Biology*, **11** (1999), 190–196.
15. Nijhawan, D., Honarpour, N., Wang, X. Apotosis in neural development and disease. *Annual Reviews Neuroscience*, **23** (2000), 73–87.
16. Oltavi, Z.N., Milliman, C.L., Korsmeyer, S.J. Bcl-2 heterodimerizes in vivo with a conserved homolog, Bax, that accelerates programmed cell death. *Cell*, **74**, 4 (1993), 609–619.
17. Gh. Păun, Computing with membranes. *Journal of Computer and System Sciences*, **61**, 1 (2000), 108–143, and *Turku Center for Computer Science-TUCS Report* Nr. 208, 1998.
18. Gh. Păun, *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.

19. Gh. Păun, G. Rozenberg, A guide to membrane computing. *Theoretical Computer Science*, **287** (2002), 73–100.
20. Pérez-Jiménez, M.J., Romero-Campero, F.J. P systems, a new computationl modelling tool for systems biology, *Transactions on Computational Systems Biology VI, LNBI* **4220**, 2006, 176–197.
21. Pérez-Jiménez, M.J., Romero-Campero, F.J. A study of the robustness of the EGFR signalling cascade using continuous membrane systems. *Lecture Notes in Computer Science*, **3561** (2005), 268 – 278.
22. Regev, A., Shapiro, E. (2004) The π-calculus as an abstraction for biomolecular systems. In G. Ciobanu and G. Rozenberg, editors, *Modelling in Molecular Biology*, Springer Berlin.
23. Scaffidi, C., Fulda. S., Srinivasan, A., Friesen, C., Li, F., Tomaselli, K.J., Debatin, K.M., Krammer, P.H., Peter, M.E. Two CD95 (APO-1/Fas) signaling pathways. *The Embo Journal*, **17** (1998), 1675–1687.
24. Schoeberl, B., Eichler–Jonsson, C., Gilles, E.D., Muller, G. Computational modeling of the dynamics of the MAP kinase cascade activated by surface and internalized EGF receptors, *Nature Biotechnology*, **20**, 4 (2002), 370–375.
25. Van Kampen, N.G. *Stochastics Processes in Physics and Chemistry*, Elsevier Science B.V., Amsterdam, 1992.
26. Walker, D.C., Southgate, J., Hill, G., Holcombe, M., Hose, D.R., Wood S.M., MacNeil, S., Smallwood, R.H. The epitheliome: modelling the social behaviour of cells. *BioSystems*, **76**, 1–3 (2004), 89–100.
27. Wang, K., Yin, X.M., Chao, D.T., Milliman, C.L., Korsmeyer, S.J. BID: a novel BH3 domain-only death agonist. *Genes & Development*, **10** (1996), 2859–2869.
28. Wells, A. EGFR–receptor. *Int. Journal Biochem. Cell Biology*, **31** (1999), 637–643.
29. Wiley, H.S., Shvartsman, S.Y., Lauffenburger, D.A. Computational modeling of the EGFR–receptor system: A paradigm for systems biology. *Trends in Cell Biology*, **13**, 1 (2003), 43–50.
30. ISI web page: `http://esi-topics.com/erf/october2003.html`
31. SciLab Web Site `http://scilabsoft.inria.fr/`
32. P Systems Modelling Framework Web Site: `http://www.dcs.shef.ac.uk/~marian/PSimulatorWeb/P_Systems_applications.htm`

Extended Spiking Neural P Systems

Artiom Alhazov[1,2], Rudolf Freund[3],
Marion Oswald[3], and Marija Slavkovik[3]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD 2028, Moldova
artiom@math.md
[2] Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
artiome.alhazov@estudiants.urv.cat
[3] Faculty of Informatics
Vienna University of Technology
Favoritenstr. 9, A-1040 Wien, Austria
{rudi,marion,marija}@emcc.at

Abstract. We consider extended variants of spiking neural P systems and show how these extensions of the original model allow for easy proofs of the computational completeness of extended spiking neural P systems and for the characterization of semilinear sets and regular languages by finite extended spiking neural P systems (defined by having only finite checking sets in the rules assigned to the cells) with only a bounded number of neurons.

1 Introduction

Just recently, a new variant of P systems was introduced based on the biological background of neurons sending electrical impulses along axons to other neurons. This biological background had already led to several models in the area of neural computation, e.g., see [11], [12], and [8]. In the area of P systems, one basic model considers hierarchical membrane structures, whereas in another important model cells are placed in the nodes of a graph (which variant was first considered in [18]; tissue P systems then were further elaborated, for example, in [7] and [13]). Based on the structure of this model of tissue P systems, in [10] the new model of spiking neural P systems was introduced. The reader is referred to this seeding paper for the interesting details of the biological motivation for this kind of P systems; we will recall just a few of the most important features:

In spiking neural P systems, the contents of a cell (neuron) consists of a number of so-called *spikes*. The rules assigned to a cell allow us to send information to other neurons in the form of electrical impulses (also called spikes) which are summed up at the target cell; the application of the rules depends on the contents of the neuron and in the general case is described by regular sets. As inspired from biology, the cell sending out spikes may be "closed" for a specific

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 123–134, 2006.

time period corresponding to the refraction period of a neuron; during this refraction period, the neuron is closed for new input and cannot get excited ("fire") for spiking again.

The length of the axon may cause a time delay before a spike arrives at the target. Moreover, the spikes coming along different axons may cause effects of different magnitude. We shall include these features in our extended model of spiking neural P systems considered below. Some other features also motivated from biology will shortly be discussed in Section 5, e.g., the use of inhibiting neurons or axons, respectively. From a mathematical point of view, the most important theoretical feature we shall include in our model of extended spiking neural P systems is that we allow the neurons to send spikes along the axons with different magnitudes at different moments of time.

In [10] the output of a spiking neural P system was considered to be the time between two spikes in a designated output cell. It was shown how spiking neural P systems in that way can generate any recursively enumerable set of natural numbers. Moreover, a characterization of semilinear sets was obtained by spiking neural P system with a bounded number of spikes in the neurons. These results can also be obtained with even more restricted forms of spiking neural P systems, e.g., no time delay (refraction period) is needed, as it was shown in [9]. In [17], the behavior of spiking neural P systems on infinite strings and the generation of infinite sequences of 0 and 1 (the case when the output neuron spikes) was investigated. Finally, in [1], the generation of strings (over the binary alphabet 0 and 1) by spiking neural P systems was investigated; due to the restrictions of the original model of spiking neural P systems, even specific finite languages cannot be generated, but on the other hand, regular languages can be represented as inverse-morphic images of languages generated by finite spiking neural P systems, and even recursively enumerable languages can be characterized as projections of inverse-morphic images of languages generated by spiking neural P systems. The problems occurring in the proofs are also caused by the quite restricted way the output is obtained from the output neuron as sequence of symbols 0 and 1. The strings of a regular or recursively enumerable language could be obtained directly by collecting the spikes sent by specific output neurons for each symbol.

In the extended model introduced in this paper, we shall use a specific output neuron for each symbol. Computational completeness can be obtained by simulating register machines as in the proofs elaborated in the papers mentioned above, yet in an easier way using only a bounded number of neurons. Moreover, regular languages can be characterized by finite extended spiking neural P systems; again, only a bounded number of neurons is really needed.

The rest of the paper is organized as follows: In the next section, we recall some preliminary notions and definitions, especially the definition and some well-known results for register machines. In section 3 we define our extended model of spiking neural P systems and explain how it works. The generative power of extended spiking neural P systems is investigated in section 4. Finally, in section 5 we give a short summary of the results obtained in this paper and

discuss some further variants of extended spiking neural P systems, especially variants with inhibiting neurons or axons.

2 Preliminaries

For the basic elements of formal language theory needed in the following, we refer to any monograph in this area, in particular, to [2] and [19]. We just list a few notions and notations: V^* is the free monoid generated by the alphabet V under the operation of concatenation and the empty string, denoted by λ, as unit element; for any $w \in V^*$, $|w|$ denotes the number of symbols in w (the *length* of w). $\mathbb{N}_+$ denotes the set of positive integers (natural numbers), $\mathbb{N}$ is the set of non-negative integers, i.e., $\mathbb{N} = \mathbb{N}_+ \cup \{0\}$. The interval of non-negative integers between k and m is denoted by $[k..m]$. Observe that there is a one-to-one correspondence between a set $M \subseteq \mathbb{N}$ and the one-letter language $L(M) = \{a^n \mid n \in M\}$; e.g., M is a regular (semilinear) set of non-negative integers if and only if $L(M)$ is a regular language. By $FIN\left(\mathbb{N}^k\right)$, $REG\left(\mathbb{N}^k\right)$, and $RE\left(\mathbb{N}^k\right)$, for any $k \in \mathbb{N}$, we denote the sets of subsets of $\mathbb{N}^k$ that are finite, regular, and recursively enumerable, respectively.

By $REG\ (REG(V))$ and $RE\ (RE(V))$ we denote the family of regular and recursively enumerable languages (over the alphabet V, respectively). By $\Psi_T(L)$ we denote the Parikh image of the language $L \subseteq T^*$, and by $PsFL$ we denote the set of Parikh images of languages from a given family FL. In that sense, $PsRE(V)$ for a k-letter alphabet V corresponds with the family of recursively enumerable sets of k-dimensional vectors of non-negative integers.

2.1 Register Machines

The proofs of the results establishing computational completeness in the area of P systems often are based on the simulation of register machines; we refer to [14] for original definitions, and to [4] for definitions like those we use in this paper:

An *n-register machine* is a construct $M = (n, P, l_0, l_h)$, where n is the number of registers, P is a finite set of instructions injectively labeled with elements from a given set $Lab(M)$, l_0 is the initial/start label, and l_h is the final label.

The instructions are of the following forms:

- $l_1 : (A(r), l_2, l_3)$ (ADD instruction)
  Add 1 to the contents of register r and proceed to one of the instructions (labeled with) l_2 and l_3.
- $l_1 : (S(r), l_2, l_3)$ (SUB instruction)
  If register r is not empty, then subtract 1 from its contents and go to instruction l_2, otherwise proceed to instruction l_3.
- $l_h : halt$ (HALT instruction)
  Stop the machine. The final label l_h is only assigned to this instruction.

A (non-deterministic) register machine M is said to generate a vector $(s_1, \ldots, s_\beta)$ of natural numbers if, starting with the instruction with label l_0 and all registers

containing the number 0, the machine stops (it reaches the instruction $l_h : halt$) with the first β registers containing the numbers $s_1, \ldots, s_\beta$ (and all other registers being empty).

Without loss of generality, in the succeeding proofs we will assume that in each ADD instruction $l_1 : (A(r), l_2, l_3)$ and in each SUB instruction $l_1 : (S(r), l_2, l_3)$ the labels l_1, l_2, l_3 are mutually distinct (for a short proof see [7]).

The register machines are known to be computationally complete, equal in power to (non-deterministic) Turing machines: they generate exactly the sets of vectors of non-negative integers which can be generated by Turing machines, i.e., the family $PsRE$.

The results proved in [5] (based on the results established in [14]) and [4], [6] immediately lead to the following result:

Proposition 1. *For any recursively enumerable set $L \subseteq \mathbb{N}^\beta$ of vectors of non-negative integers there exists a non-deterministic $(\beta + 2)$-register machine M generating L in such a way that, when starting with all registers 1 to $\beta + 2$ being empty, M non-deterministically computes and halts with n_i in registers i, $1 \leq i \leq \beta$, and registers $\beta+1$ and $\beta+2$ being empty if and only if $(n_1, ..., n_\beta) \in L$. Moreover, the registers 1 to β are never decremented.*

When considering the generation of languages, we can use the model of a *register machine with output tape*, which also uses a tape operation:

- $l_1 : (write(a), l_2)$
  Write symbol a on the output tape and go to instruction l_2.

We then also specify the output alphabet T in the description of the register machine with output tape, i.e., we write $M = (n, T, P, l_0, l_h)$.

The following result is folklore, too (e.g., see [14] and [3]):

Proposition 2. *Let $L \subseteq T^*$ be a recursively enumerable language. Then L can be generated by a register machine with output tape with 2 registers. Moreover, at the beginning and at the end of a successful computation generating a string $w \in L$ both registers are empty, and finally, only successful computations halt.*

3 Extended Spiking Neural P Systems

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [15]; comprehensive information can be found on the P systems web page `http://psystems.disco.unimib.it`. Moreover, for the motivation and the biological background of spiking neural P systems we refer the reader to [10].

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 *cells* (or *neurons*); the neurons are uniquely identified by a number between 1 and m (obviously, we could instead use an alphabet with m symbols to identify the neurons);
- S describes the *initial configuration* by assigning an initial value (of spikes) to each neuron; for the sake of simplicity, we assume that at the beginning of a computation we have no pending packages along the axons between the neurons;
- R is a finite set of *rules* of the form $\big(i, E/a^k \to P; d\big)$ such that $i \in [1..m]$ (specifying that this rule is assigned to cell 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, d is the *delay* (the "refraction time" when neuron i performs this rule), and P is a (possibly empty) set of *productions* of the form (l, w, t) where $l \in [1..m]$ (thus specifying the target cell), $w \in \{a\}^*$ is the *weight* of the energy sent along the axon from neuron i to neuron l, and t is the time needed before the information sent from neuron i arrives at neuron l (i.e., the *delay along the axon*). If the checking sets in all rules are finite, then Π is called a *finite ESNP system*.

A *configuration* of the ESNP system is described as follows:

- for each neuron, the actual number of spikes in the neuron is specified;
- in each neuron i, we may find an "activated rule" $\big(i, E/a^k \to P; d'\big)$ waiting to be executed where d' is the remaining time until the neuron spikes;
- in each axon to a neuron l, we may find pending packages of the form (l, w, t') where t' is the remaining time until $|w|$ spikes have to be added to neuron l provided it is not closed for input at the time this package arrives.

A *transition* from one configuration to another one now works as follows:

- for each neuron i, we first check whether we find an "activated rule" $\big(i, E/a^k \to P; d'\big)$ waiting to be executed; if $d' = 0$, then neuron i "spikes", i.e., for every production (l, w, t) occurring in the set P we put the corresponding package (l, w, t) on the axon from neuron i to neuron l, and after that, we eliminate this "activated rule" $\big(i, E/a^k \to P; d'\big)$;
- for each neuron l, we now consider all packages (l, w, t') on axons leading to neuron l; provided the neuron is not closed, i.e., if it does not carry an activated rule $\big(i, E/a^k \to P; d'\big)$ with $d' > 0$, we then sum up all weights w in such packages where $t' = 0$ and add this sum of spikes to the corresponding number of spikes in neuron l; in any case, the packages with $t' = 0$ are eliminated from the axons, whereas for all packages with $t' > 0$, we decrement t' by one;
- for each neuron i, we now again check whether we find an "activated rule" $\big(i, E/a^k \to P; d'\big)$ (with $d' > 0$) or not; if we have not found an "activated rule", we now may apply any rule $\big(i, E/a^k \to P; d\big)$ from R for which the

current number of spikes in the neuron is in E and then put a copy of this rule as "activated rule" for this neuron into the description of the current configuration; on the other hand, if there still has been an "activated rule" $(i, E/a^k \to P; d')$ in the neuron with $d' > 0$, then we replace d' by $d' - 1$ and keep $(i, E/a^k \to P; d' - 1)$ as the "activated rule" in neuron i in the description of the configuration for the next step of the computation.

After having executed all the substeps described above in the correct sequence, we obtain the description of the new configuration. A *computation* is a sequence of configurations starting with the initial configuration given by S. A computation is called *successful* if it halts, i.e., if no pending package can be found along any axon, no neuron contains an activated rule, and for no neuron, a rule can be activated.

In the original model introduced in [10], in the productions (l, w, t) of a rule $(i, E/a^k \to \{(l, w, t)\}; d)$, only $w = a$ (for *spiking rules*) or $w = \lambda$ (for *forgetting rules*) as well as $t = 0$ was allowed (and for forgetting rules, the checking set E had to be finite and disjoint from all other sets E in rules assigned to neuron i). Moreover, reflexive axons, i.e., leading from neuron i to neuron i, were not allowed, hence, for (l, w, t) being a production in a rule $(i, E/a^k \to P; d)$ for neuron i, $l \neq i$ was required. Yet the most important extension is that different rules for neuron i may affect different axons leaving from it whereas in the original model the structure of the axons (called synapses there) was fixed. Finally, we should like to mention that the sequence of substeps leading from one configuration to the next one together with the interpretation of the rules from R was taken in such a way that the original model can be interpreted in a consistent way within the extended model introduced in this paper. From a mathematical point of view, another interpretation in our opinion would have been more suitable: whenever a rule $(i, E/a^k \to P; d)$ is activated, the packages induced by the productions (l, w, t) in the set P of a rule $(i, E/a^k \to P; d)$ activated in a computation step are immediately put on the axon from neuron i to neuron l, whereas the delay d only indicates the refraction time for neuron i itself, i.e., the time period this neuron will be closed. Yet as in the proofs of computational completeness given below we shall not need any of the delay features, we shall not investigate this variant in more details anymore in the rest of the paper.

Depending on the purpose the ESNP system shall be used, some more features have to be specified: for generating k-dimensional vectors of non-negative integers, we have to designate k neurons as *output neurons*; the other neurons then will also be called *actor neurons*. Without loss of generality, in the following we shall assume the output neurons to be the first k neurons of the ESNP system. Moreover, for the sake of conciseness, we shall also assume that no rules are assigned to these output neurons (in the original model they correspond to a sensor in the environment of the system; in some sense, they are not neurons of the system itself). There are several possibilities to define how the output values are computed; according to [10], we can take the distance between the first two spikes in an output neuron to define its value; in this paper, we shall prefer to take the number of spikes at the end of a successful computation in the

neuron as the output value. For generating strings, we do not interpret the spike train of a single output neuron as done, for example, in [1], but instead consider the sequence of spikes in the output neurons each of them corresponding to a specific terminal symbol; if more than one output neuron spikes, we take any permutation of the corresponding symbols as the next substring of the string to be generated.

The delay t in productions (l, w, t) can be used to replace the delay in the neurons themselves in many of the constructions elaborated, for example, in [10], [16], and [1]; there often a subconstruction is implemented which ensures that a neuron l_3 gets a spike one time step later than a neuron l_2, both getting the impulse from a neuron l_1; to accomplish this task in the original model, two intermediate neurons are needed using the refraction period delay of one neuron, whereas we can get this effect directly from neuron l_1 by using the delay along the axons using the rule $\left(l_1, E/a^k \to \{(l_2, a, 0), (l_3, a, 1)\}; 0\right)$. In that way, only this feature allows for simpler proofs; on the other hand, taking into account the other extensions in ESNP systems as defined above, we shall not need any of the delay features for the proofs of computational completeness given below.

4 ESNP Systems as Generating Devices

We now consider extended spiking neural P systems as generating devices. As throughout this section we do not use delays in the rules and productions, we simply omit them to keep the description of the systems concise, i.e., for a production (i, w, t) we simply write (i, w); for example, instead of $\left(2, \left\{a^i\right\}/a^i \to \left\{(1, a, 0), \left(2, a^j, 0\right)\right\}; 0\right)$ we write $\left(2, \left\{a^i\right\}/a^i \to \left\{(1, a), \left(2, a^j\right)\right\}\right)$.

The following example gives a characterization of regular sets of non-negative integers:

Example 1. Any semilinear set of non-negative integers M can be generated by a finite ESNP system with only two neurons.

Let M be a semilinear set of non-negative integers and consider a regular grammar G generating the language $L(G) \subseteq \{a\}^*$ with $N(L(G)) = M$; without loss of generality we assume the regular grammar to be of the form $G = (N, \{a\}, A_1, P)$ with the set of non-terminal symbols N, $N = \{A_i \mid 1 \le i \le m\}$, the start symbol A_1, and P the set of regular productions of the form $B \to aC$ with $B, C \in N$ and $A \to \lambda$. We now construct the finite ESNP system $\Pi = (2, S, R)$ that generates an element of M by the number of spikes contained in the output neuron 1 at the end of a halting computation: we start with one spike in neuron 2 (representing the start symbol A_1 and no spike in the output neuron 1, i.e., $S = \{(1, 0), (2, 1)\}$. The production $A_i \to aA_j$ is simulated by the rule $\left(2, \{i\}/a^i \to \left\{(1, a), \left(2, a^j\right)\right\}\right)$ and $A_i \to \lambda$ is simulated by the rule $\left(2, \{i\}/a^i \to \emptyset\right)$, i.e., in sum we obtain

$$
\begin{aligned}
\Pi &= (2, S, R), \\
S &= \{(1, 0), (2, 1)\}, \\
R &= \left\{\left(2, \{i\}/a^i \to \left\{(1, a), \left(2, a^j\right)\right\}\right) \mid 1 \le i, j \le m, A_i \to aA_j \in P\right\} \\
&\quad \cup \left\{\left(2, \{i\}/a^i \to \emptyset\right) \mid 1 \le i \le m, A_i \to \lambda \in P\right\}.
\end{aligned}
$$

Neuron 2 keeps track of the actual non-terminal symbol and stops the derivation as soon as it simulates a production $A_i \to \lambda$, because finally neuron 2 is empty. In order to guarantee that this is the only way how we can obtain a halting computation in Π, without loss of generality we assume G to be reduced, i.e., for every non-terminal symbol A from N there is a regular production with A on the left-hand side. These observations prove that we have $N(L(G)) = M$.

We can also generate the numbers in M as the difference between the (first) two spikes arriving in the output neuron by the following ESNP system Π':

$$\begin{aligned}
\Pi' &= (2, S', R'), \\
S' &= \{(1,0), (2, m+1)\}, \\
R' &= \left\{\left(2, \{i\}/a^i \to \left\{\left(2, a^j\right)\right\}\right) \mid 1 \le i, j \le m, A_i \to aA_j \in P\right\} \\
&\quad \cup \left\{\left(2, \{i\}/a^i \to \{(1,a)\}\right) \mid 1 \le i \le m, A_i \to \lambda \in P\right\} \\
&\quad \cup \left\{\left(2, \{m+1\}/a^{m+1} \to \{(1,a),(2,a)\}\right)\right\}.
\end{aligned}$$

We should like to mention that one reason for the constructions given above to be that easy is the fact that we allow "reflexive" axons, i.e., we can keep track of the actual non-terminal symbol without delay. We could avoid this by adding an additional neuron 3 thus obtaining the following finite ESNP system:

$$\begin{aligned}
\Pi'' &= (3, S'', R''), \\
S'' &= \{(1,0), (2,0), (3,1)\}, \\
R'' &= \left\{\left(2, \left\{a^i\right\}/a^i \to \left\{\left(3, a^j\right), (1,a)\right\}\right), \left(3, \left\{a^i\right\}/a^i \to \left\{\left(2, a^i\right)\right\}\right)\right. \\
&\qquad \left. \mid 1 \le i, j \le m, A_i \to aA_j \in P\right\} \\
&\quad \cup \left\{\left(2, \left\{a^i\right\}/a^i \to \emptyset\right) \mid 1 \le i \le m, A_i \to \lambda \in P\right\}.
\end{aligned}$$

Observe that the derivation in the corresponding grammar now is delayed by a factor of 2 in the computation in Π, because we need one step to propagate the information from neuron 3 to neuron 2 which then sends the spikes to the output neuron. Hence, an interpretation of the generated number as the difference between two spikes in the output neuron is not possible anymore.

Lemma 1. *For any ESNP system where during a computation only a bounded number of spikes occurs in the actor neurons, the generated language is regular.*

Proof (sketch). Let Π be an ESNP system where during a computation only a bounded number of spikes occurs in the actor neurons. Then the number of configurations differing in the actor neurons and the packages along the axons, but without considering the contents of the output neurons, is finite, hence, we can assign non-terminal symbols A_k to each of these configurations and take all right-regular productions $A_i \to wA_j$ such that w is a permutation of the string obtained by concatenating the symbols indicated by the number of added spikes when going from configuration i to configuration j. If we interpret just the number of spikes in the output neurons as the result of a successful computation, then we obtain the Parikh set of the language, which then obviously is regular, i.e., semilinear. □

As we shall see later in this section, ESNP systems are computationally complete, hence, the question whether in an ESNP system during a computation only a

bounded number of spikes occurs in the actor neurons is not decidable, but there is a simple syntactic feature that allows us to characterize regular sets, namely the finiteness of an ESNP system. The following theorem is a consequence of the preceding lemma and the example given above:

Theorem 1. *Any regular language L with $L \subseteq T^*$ for a terminal alphabet T with $card\,(T) = n$ can be generated by a finite ESNP system with $n+1$ neurons. On the other hand, every language generated by a finite ESNP system is regular.*

Proof. Let G be a regular grammar generating the language $L\,(G) = L \subseteq T^*$, $T = \{a_k \mid 1 \leq k \leq n\}$; without loss of generality we assume the regular grammar to be of the form $G = (N, T, A_1, P)$ with the set of non-terminal symbols N, $N = \{A_i \mid 1 \leq i \leq m\}$, the start symbol A_1, and P the set of regular productions of the form $A_i \to a_k A_j$ with $A_i, A_j \in N$, $a_k \in T$, and $A_i \to \lambda$ with $A_i \in N$. We now construct the finite ESNP system $\Pi = (n+1, S, R)$ that generates an element of L by the sequence of spikes in the output neurons 1 to n corresponding with the desired string during a halting computation: we start with one spike in neuron $n+1$ (representing the start variable A_1) and no spike in the output neurons 1 to n:

$$\begin{aligned}
\Pi &= (n+1, S, R)\,,\\
S &= \{(1,0)\,, ..., (n,0)\,, (n+1,1)\}\,,\\
R &= \left\{\left(n+1, \left\{a^i\right\}/a^i \to \left\{(k,a)\,, \left(n+1, a^j\right)\right\}\right)\right.\\
&\quad \left. \mid 1 \leq i,j \leq m, 1 \leq k \leq n, A_i \to a_k A_j \in P\right\}\\
&\quad \cup \left\{\left(n+1, \left\{a^i\right\}/a^i \to \emptyset\right) \mid 1 \leq i \leq m, A_i \to \lambda \in P\right\}.
\end{aligned}$$

Obviously, $L\,(G) = L\,(\Pi) = L$.

On the other hand, let $\Pi = (m, S, R)$ be a finite ESNP system. Then from Π we can construct an equivalent finite ESNP system $\Pi' = (m, S, R')$ such that Π' fulfills the requirements of Lemma 1: let x be the maximal number occurring in all the checking sets of rules from R and let y be (a number not less than) the maximal number of spikes that can be added in one computation step to any of the m neurons of Π (an upper bound for y is the maximal number of weights in the productions of the rules in R multiplied by the maximal delay in these productions $+1$ multiplied by the maximal number of axons ending in a neuron of Π); then define

$$R' = R \cup \left\{\left(i, \left\{a^{x+1+k}\right\}/a^k \to \emptyset\right) \mid 1 \leq k < 2y\right\}.$$

Hence, the maximal number of spikes in any of the neurons of Π' is $x + 2y$, therefore Lemma 1 can be applied (observe that the additional rules in R' cannot lead to additional infinite computations, because they only consume spikes, but let the contents of the neurons stay above x, hence, no other rules become applicable). □

Corollary 1. *Any semilinear set of n-dimensional vectors can be generated by a finite ESNP system with $n+1$ neurons. On the other hand, every set of n-dimensional vectors generated by a finite ESNP system is semilinear.*

Proof. The result directly follows from Theorem 1 by just taking the number of spikes in the output neurons, i.e., the corresponding Parikh set of the generated language (because $PsREG(\{a_i \mid 1 \leq i \leq n\}) = REG(\mathbb{N}^n)$). □

We now show that every recursively enumerable language over an n-letter alphabet can be generated by an ESNP system with a bounded number of neurons:

Theorem 2. *Any recursively enumerable language L with $L \subseteq T^*$ for a terminal alphabet T with $card(T) = n$ can be generated by an ESNP system with $n + 2$ neurons.*

Proof. Let $M = (d, T, P, l_0, l_h)$ be a register machine with output tape generating L (according to Proposition 2, $d = 2$ is sufficient), and without loss of generality, let $Lab(M) = [0..m-1]$, $l_0 = 1$ (the start label), and $l_h = 0$ (the final label). Then we construct an ESNP system $\Pi = (n + d, S, R)$ as follows:

$$
\begin{aligned}
\Pi &= (n+d, S, R), \\
S &= \{(i, \lambda) \mid 1 \leq i \leq n+d, i \neq n+1\} \cup \{(n+1, a)\}, \\
R &= \left\{ \left(n+1, \left\{a^{mj+l} \mid j \in \mathbb{N}\right\} / a^l \rightarrow \left\{\left(n+1, a^{m+k}\right)\right\}\right) \right. \\
&\quad \left. \mid l : (A(n+1), l', l'') \in P, k \in \{l', l''\} \right\} \\
&\quad \cup \left\{ \left(n+1, \left\{a^{mj+l} \mid j \in \mathbb{N}\right\} / a^l \rightarrow \left\{\left(n+1, a^k\right), \left(n+i, a^m\right)\right\}\right) \right. \\
&\quad \left. \mid l : (A(n+i), l', l'') \in P, k \in \{l', l''\}, 1 < i \leq d \right\} \\
&\quad \cup \left\{ \left(n+1, \left\{a^l\right\} / a^l \rightarrow \left\{\left(n+1, a^{l''}\right)\right\}\right), \right. \\
&\quad \left. \left(n+1, \left\{a^{mj+l} \mid j \in \mathbb{N}_+\right\} / a^{l+m} \rightarrow \left\{\left(n+1, a^{l'}\right)\right\}\right) \right. \\
&\quad \left. \mid l : (S(n+1), l', l'') \in P \right\} \\
&\quad \cup \left\{ \left(n+1, \left\{a^{mj+l} \mid j \in \mathbb{N}\right\} / a^l \rightarrow \left\{\left(n+i, a^l\right)\right\}\right), \right. \\
&\quad \left(n+i, \left\{a^l\right\} / a^l \rightarrow \left\{\left(n+1, a^{l''}\right)\right\}\right), \\
&\quad \left(n+i, \left\{a^{mj+l} \mid j \in \mathbb{N}_+\right\} / a^{l+m} \rightarrow \left\{\left(n+1, a^{l'}\right)\right\}\right), \\
&\quad \left. \mid l : (S(n+i), l', l'') \in P, 1 < i \leq d \right\} \\
&\quad \cup \left\{ \left(n+1, \left\{a^{mj+l} \mid j \in \mathbb{N}\right\} / a^l \rightarrow \left\{\left(n+1, a^{l'}\right), (s, a)\right\}\right), \right. \\
&\quad \left. \mid l : (write(a_s), l') \in P, 1 \leq s \leq n \right\}.
\end{aligned}
$$

The neurons 1 to n are the output neurons and the actor neurons $n + i$, $1 \leq i \leq d$, represent the d registers of M. The contents c_i of register i of M is stored as mc_i spikes in neuron $n + i$.

To simulate the instructions of P, the label l of the current instruction is also stored in neuron $n + 1$, which then contains $mc_1 + l$ spikes and thus guides the whole computation. Whenever a SUB instruction on a register $i > 1$ has to be simulated, in an intermediate step the control temporarily goes to register i which then contains $mc_i + l$. A tape operation $l : (write(a_k), l')$ is simulated by sending a spike to the output neuron representing symbol a_k.

As at the end of a successful computation all registers are empty and $l_h = 0$, also the computation in Π stops because the actor neurons $n+i$, $1 \leq i \leq d$, will not spike anymore. □

Corollary 2. *Any recursively enumerable set of n-dimensional vectors can be generated by an ESNP system with $n+2$ neurons.*

Proof. The result directly can be proved by using Proposition 1 and a similar construction as that one elaborated in the proof of Theorem 2, yet it also follows from Theorem 2 by just taking the number of spikes in the output neurons as the value of the components of the n-dimensional vector, i.e., by taking the corresponding Parikh set of the generated language (because $PsRE(\{a_i \mid 1 \leq i \leq n\}) = RE(\mathbb{N}^n)$). □

5 Summary and Further Variants

In this paper, we have considered various extensions of the original model of spiking neural P systems, some of them arising from biological motivations, some others being more of mathematical interest than having a biological interpretation. The extensions considered in more detail here allowed for establishing computational completeness in an easy way, and moreover, we got a quite natural characterization of semilinear sets of (vectors of) non-negative integers and regular languages, respectively, by finite extended spiking neural P systems with a bounded number of neurons. On the other hand, in the future some other restrictions should be investigated allowing for the characterization of sets in a family between the families of regular and recursively enumerable sets (of vectors of non-negative integers or strings).

A quite natural feature found in biology and also used in the area of neural computation is that of inhibiting neurons or axons between neurons. We can include this feature in our extended model of spiking neural P systems considered above in a variant closely related to the original model of spiking neural P systems by specifying certain connections from one neuron to another one as inhibiting ones – the spikes coming along such inhibiting axons then would close the target neuron for a time period given by the sum of all inhibiting spikes.

Acknowledgements

The first and the second author very much appreciate the interesting discussions with Gheorghe Păun during the Brainstorming Week on Membrane Computing 2006 in Sevilla on the main features of spiking neural P systems. The work of Artiom Alhazov is partially supported by the project TIC2003-09319-C03-01 from Rovira i Virgili University. The work of Marion Oswald is supported by FWF-project T225-N04.

References

1. Chen H, Freund R, Ionescu M, Păun Gh, Pérez-Jiménez MJ (2006) On String Languages Generated by Spiking Neural P Systems. In: Gutiérrez-Naranjo MA, Păun Gh, Riscos-Núñez A, Romero-Campero FJ (eds) Fourth Brainstorming Week on Membrane Computing, Vol. I RGNC REPORT 02/2006, Research Group on Natural Computing, Sevilla University, Fénix Editora, 169–194
2. Dassow J, Păun Gh (1989) Regulated Rewriting in Formal Language Theory. Springer, Berlin
3. Fernau H, Freund R, Oswald M, Reinhardt K (2005) Refining the Nonterminal Complexity of Graph-controlled Grammars. In: Mereghetti C, Palano B, Pighizzini G, Wotschke D (eds) Seventh International Workshop on Descriptional Complexity of Formal Systems, 110–121
4. Freund R, Oswald M (2003) P Systems with activated/prohibited membrane channels. In: Păun Gh , Rozenberg G, Salomaa A, Zandron C (eds) Membrane Computing. International Workshop WMC 2002, Curtea de Argeş, Romania. Lecture Notes in Computer Science 2597, Springer, Berlin, 261–268.
5. Freund R, Oswald M (2002) GP Systems with Forbidding Context. Fundamenta Informaticae 49:81–102
6. Freund R, Păun Gh (2004) From Regulated Rewriting to Computing with Membranes: Collapsing Hierarchies. Theoretical Computer Science 312:143–188
7. Freund R, Păun Gh, Pérez-Jiménez M J (2004) Tissue-like P systems with channel states. Theoretical Computer Science 330:101–116
8. Gerstner W, Kistler W (2002) Spiking Neuron Models. Single Neurons, Populations, Plasticity. Cambridge Univ. Press
9. Ibarra OH, Păun A, Păun Gh, Rodríguez-Patón A, Sosík P, Woodworth S (2006) Normal Forms for Spiking Neural P Systems. In: Gutiérrez-Naranjo MA, Păun Gh, Riscos-Núñez A, Romero-Campero FJ (eds) Fourth Brainstorming Week on Membrane Computing, Vol. II RGNC REPORT 02/2006, Research Group on Natural Computing, Sevilla University, Fénix Editora, 105–136
10. Ionescu M, Păun Gh, Yokomori T (2006) Spiking neural P systems. Fundamenta Informaticae 71, 2–3:279–308
11. Maass W (2002) Computing with spikes. Special Issue on Foundations of Information Processing of TELEMATIK 8, 1:32–36
12. Maass W, Bishop C (eds) (1999) Pulsed Neural Networks. MIT Press, Cambridge
13. Martín-Vide C, Pazos J, Păun Gh, Rodríguez-Patón A (2002) A new class of symbolic abstract neural nets: Tissue P systems. In: Proceedings of COCOON 2002, Singapore, Lecture Notes in Computer Science 2387, Springer-Verlag, Berlin, 290–299
14. Minsky M L (1967) Computation: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, New Jersey
15. Păun Gh (2002) Computing with Membranes: An Introduction. Springer, Berlin
16. Păun Gh, Pérez-Jiménez MJ, Rozenberg G (2006) Spike trains in spiking neural P systems, Intern J Found Computer Sci, to appear (also available at [20])
17. Păun Gh, Pérez-Jiménez MJ, Rozenberg G (2006) Infinite spike trains in spiking neural P systems. Submitted
18. Păun Gh, Sakakibara Y, Yokomori T (2006) P systems on graphs of restricted forms. Publicationes Mathematicae Debrecen 60:635–660
19. Rozenberg G, Salomaa A (eds) (1997) Handbook of Formal Languages (3 volumes). Springer, Berlin
20. The P Systems Web Page, `http://psystems.disco.unimib.it`

Towards a Characterization of P Systems with Minimal Symport/Antiport and Two Membranes*

Artiom Alhazov[1,2] and Yurii Rogozhin[1]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
{artiom,rogozhin}@math.md
[2] Research Group on Mathematical Linguistics
Rovira i Virgili University, Tarragona, Spain
artiome.alhazov@estudiants.urv.cat

Abstract. We prove that any set of numbers containing zero generated by symport/antiport P systems with two membranes and minimal cooperation is finite (for both symport/antiport P systems and for purely symport P systems). On the other hand, one additional object in the output membrane allows symport/antiport P systems (purely symport P systems) with two membranes and minimal cooperation generate any recursively enumerable sets of natural numbers without zero. Thus we improve our previous results for symport/antiport P systems with two membranes and minimal cooperation from three "garbage" objects down to one object and for purely symport P systems from six objects down to one object. Thus we show the optimality of these results.

1 Introduction

P systems with *symport/antiport* rules, i.e., P systems with *pure communication rules assigned to membranes*, first were introduced in [19]; symport rules move objects across a membrane together in one direction, whereas antiport rules move objects across a membrane in opposite directions. These operations are very powerful, i.e., P systems with symport/antiport rules have universal computational power with only one membrane, e.g., see [10], [14], [11].

A comprehensive overview of the most important results obtained in the area of P systems and tissue P systems with *symport/antiport* rules (with respect to the development of computational completeness results improving descriptional complexity parameters as the number of membranes and cells, respectively, the weight of the rules and the number of objects) can be found in [1].

In this paper, we first show that if some P system with two membranes and with minimal cooperation, i.e., a P system with symport/antiport rules of weight one or a P system with symport rules of weight two, generates a set of numbers

* The authors acknowledge the project 06.411.03.04P from the Supreme Council for Science and Technological Development of the Academy of Sciences of Moldova.

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 135–153, 2006.

containing zero, then this set is **finite**. After that we prove that P systems with symport/antiport rules of weight one can generate any *recursively enumerable* set of natural numbers without zero (i.e., they are computationally complete with just **one superfluous object** remaining in the output membrane at the end of a halting computation). The same result is true also for purely symport P systems of weight two. In this way, we improve the results from [1] for symport/antiport P systems with two membranes and minimal cooperation from three "'garbage" objects down to one object and for purely symport P systems with two membranes and minimal cooperation from six objects down to one object. Thus we show the optimality of these results.

Notice that symport/antiport P systems with three membranes and minimal cooperation can generate any recursively enumerable sets of natural numbers without using superfluous objects in the output membrane [3]. The question about precise characterization of computational power of symport/antiport P systems (purely symport P systems) with two membranes and minimal cooperation is still open.

2 Basic Notations and Definitions

For the basic elements of formal language theory needed in the following, we refer to [24]. We just list a few notions and notations: $\mathbb{N}$ denotes the set of natural numbers (i.e., of non-negative integers). $V*$ is the free monoid generated by the alphabet V under the operation of concatenation and the empty string, denoted by λ, as unit element; by $\mathbb{N}RE$, $\mathbb{N}REG$, and $\mathbb{N}FIN$ we denote the family of recursively enumerable sets, regular sets, and finite sets of natural numbers, respectively. For $k \geq 1$, by $\mathbb{N}_k RE$ we denote the family of recursively enumerable sets of natural numbers excluding the initial segment 0 to $k-1$. Equivalently, $\mathbb{N}_k RE = \{k + L \mid L \in \mathbb{N}RE\}$, where $k + L = \{k + n \mid n \in L\}$. Particularly, $\mathbb{N}_1 RE = \{N \in \mathbb{N}RE \mid 0 \notin N\}$. We will also use the next notations: $\mathbb{N}_{\exists 0} FIN = \{N \in \mathbb{N}FIN \mid 0 \in N\}$, $\mathbb{N}_{\exists 0} SEG_1 = \{\{k \in \mathbb{N} \mid k < n\} \mid n \geq 0\}$ and $\mathbb{N}_{\exists 0} SEG_2 = \{\{2k \mid k < n\} \mid n \geq 0\}$.

The families of recursively enumerable sets of vectors of natural numbers are denoted by $PsRE$.

2.1 Counter Automata

A non-deterministic *counter automaton* (see [9], [1]) is a construct

$$M = (d, Q, q_0, q_f, P)\,, \text{ where}$$

- d is the number of counters, and we denote $D = \{1, \ldots, d\}$;
- Q is a finite set of states, and without loss of generality, we use the notation $Q = \{q_i \mid 0 \leq i \leq f\}$ and $F = \{0, 1, \ldots, f\}$,
- $q_0 \in Q$ is the initial state,
- $q_f \in Q$ is the final state, and
- P is a finite set of instructions of the following form:

1. $(q_i \to q_l, k+)$, with $i, l \in F,\ i \neq f,\ k \in D$ (" increment" instruction). This instruction increments counter k by one and changes the state of the system from q_i to q_l.
2. $(q_i \to q_l, k-)$, with $i, l \in F,\ i \neq f,\ k \in D$ (" decrement" instruction). If the value of counter k is greater than zero, then this instruction decrements it by 1 and changes the state of the system from q_i to q_l. Otherwise (when the value of counter k is zero) the computation is blocked in state q_i.
3. $(q_i \to q_l, k = 0)$, with $i, l \in F,\ i \neq f,\ k \in D$ (" test for zero" instruction). If the value of counter k is zero, then this instruction changes the state of the system from q_i to q_l. Otherwise (the value stored in counter k is greater than zero) the computation is blocked in state q_i.
4. *halt*. This instruction stops the computation of the counter automaton, and it can only be assigned to the final state q_f.

A transition of the counter automaton consists in updating/checking the value of a counter according to an instruction of one of the types described above and by changing the current state to another one. The computation starts in state q_0 with all counters being equal to zero. The result of the computation of a counter automaton is the value of the first k counters when the automaton halts in state $q_f \in Q$ (without loss of generality we may assume that in this case all other counters are empty). A counter automaton thus (by means of all computations) generates a set of k-vectors of natural numbers.

It is known that any set of k-vectors of natural numbers from $PsRE$ can be generated by a counter automaton with $k + 2$ counters where only "increment" instructions are needed for the first k counters. We will use this in our proofs.

2.2 P Systems with Symport/Antiport Rules

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [21]; comprehensive information can be found in the P systems web page, [28].

A *P system with symport/antiport rules* is a construct

$$\Pi = (O, \mu, w_1, \ldots, w_k, E, R_1, \ldots, R_k, i_0), \text{ where}$$

1. O is a finite alphabet of symbols called *objects*;
2. μ is a *membrane structure* consisting of k membranes that are labeled in a one-to-one manner by $1, 2, \ldots, k$;
3. $w_i \in O^*$, for each $1 \leq i \leq k$, is a finite multiset of objects associated with the region i (delimited by membrane i);
4. $E \subseteq O$ is the set of objects that appear in the environment in an infinite number of copies;
5. R_i, for each $1 \leq i \leq k$, is a finite set of symport/antiport rules associated with membrane i; these rules are of the forms (x, in) and (y, out) *(symport rules)* and $(y, out; x, in)$ *(antiport rules)*, respectively, where $x, y \in O^+$;
6. i_0 is the label of an elementary membrane of μ that identifies the corresponding output region.

A P system with symport/antiport rules is defined as a computational device consisting of a set of k hierarchically nested membranes that identify k distinct regions (the membrane structure μ), where to each membrane i there are assigned a multiset of objects w_i and a finite set of symport/antiport rules R_i, $1 \leq i \leq k$. A rule $(x, in) \in R_i$ permits the objects specified by x to be moved into region i from the immediately outer region. Notice that for P systems with symport rules the rules in the skin membrane of the form (x, in), where $x \in E^*$, are forbidden. A rule $(x, out) \in R_i$ permits the multiset x to be moved from region i into the outer region. A rule $(y, out; x, in)$ permits the multisets y and x, which are situated in region i and the outer region of i, respectively, to be exchanged. It is clear that a rule can be applied if and only if the multisets involved by this rule are present in the corresponding regions. The weight of a symport rule (x, in) or (x, out) is given by $|x|$, while the weight of an antiport rule $(y, out; x, in)$ is given by $max\{|x|, |y|\}$.

As usual, a computation in a P system with symport/antiport rules is obtained by applying the rules in a non-deterministic maximally parallel manner. Specifically, in this variant, a computation is restricted to moving objects through membranes, since symport/antiport rules do not allow the system to modify the objects placed inside the regions. Initially, each region i contains the corresponding finite multiset w_i, whereas the environment contains only objects from E that appear in infinitely many copies.

A computation is successful if starting from the initial configuration, the P system reaches a configuration where no rule can be applied anymore. The result of a successful computation is a natural number that is obtained by counting all objects (only the terminal objects as it is done in [2], if in addition we specify a subset of O as the set of terminal symbols) present in region i_0. Given a P system Π, the set of natural numbers computed in this way by Π is denoted by $N(\Pi)$ (or $N(\Pi)_T$ if the terminal symbols are distinguished). If the multiplicity of each (terminal) object is counted separately, then a vector of natural numbers is obtained, denoted by $Ps(\Pi)$, see [21].

By $\mathbb{N}OP_m(sym_s, anti_t)$ we denote the family of natural number sets generated by P systems with symport/antiport rules with at most $m > 0$ membranes, symport rules of size at most $s \geq 0$, and antiport rules of size at most $t \geq 0$. By $\mathbb{N}_k OP_m(sym_s, anti_t)$ we denote the corresponding families of natural numbers without the initial segment $\{0, 1, \ldots, k-1\}$ generated by such P systems. Any unbounded parameter m, s, t is replaced by $*$. If $t = 0$, then we may omit $anti_t$.

3 The Garbage Is Unavoidable

Theorem 1. *If* $M \in \mathbb{N}OP_2(sym_1, anti_1)$, *then* $0 \in M \Rightarrow M \in \mathbb{N}FIN$.

Proof. Consider an arbitrary P system Π with two membranes and symport/antiport rules of weight one,

$$\Pi = (O, [_1 \, [_2 \ ]_2 \,]_1, w_1, w_2, E, R_1, R_2, 2).$$

For Π, consider some computation $\mathcal{C}$ generating 0: $\mathcal{C}$ ends in some configuration C with nothing in membrane 2, $u_1 \in (O-E)^*$ and $u_e \in E^*$ in membrane 1 and $u_0 \in (O-E)^*$ in the environment. Finally, consider an arbitrary halting computation $\mathcal{C}'$ of Π: $\mathcal{C}'$ ends in some configuration C' with $v_2 \in (O-E)^*$ and $v_f \in E^*$ in membrane 2, with $v_1 \in (O-E)^*$ and $v_e \in E^*$ in membrane 1 and $v_0 \in (O-E)^*$ in the environment. We are claiming that $|v_2 v_f| + |v_1 v_e| \leq |w_2| + |w_1|$ (i.e., the total number of objects in the system cannot grow without starting an infinite computation, and thus Π cannot generate numbers greater than the initial number of objects inside it).

Let us assume the contrary. Since the number of objects inside the system can only increase by symport rules, some rule $p_0 : (s_0, in) \in R_1$ had to be applied at some step of $\mathcal{C}'$ (by definition $s_0 \in O - E$). This implies that s_0 has been brought to the environment. We can assume that rules $p_i : (s_i, out; s_{i-1}, in) \in R_1$, $1 \leq i < n$, have been applied ($n \geq 0$), $s_i \in O - E$, $1 \leq i \leq n$. Suppose also that n is maximal (s_n was not brought to the environment by antiport with another object from $O - E$). Thus R_1 contains either a rule $p : (s_n, out) \in R_1$, or $p' : (s_n, out; a, in) \in R_1$, $a \in E$.

Examine the final configuration C of the computation generating 0. Recall that since region 2 is empty, we cannot "hide" anything there. If s_0 is in u_0, then p_0 can be applied, hence C is not final. Therefore (region 2 is empty) s_0 is in u_1. For all $1 \leq i \leq n$, given $s_{i-1} \in w_1$, if s_i is in u_0, then p_i can be applied, hence C is not final. Consequently (region 2 is empty), s_i is in w_1 as well. By induction, we obtain that s_n is in w_1. However, this implies that either $p \in R$ and p can be applied, or some $p' \in R$ and p' can be applied, therefore C is not final.

This implies that if a system may generate 0, then any computation where the number of objects inside the output membrane is increased cannot halt. Therefore, Π cannot generate infinite sets containing 0. □

The corresponding result also holds for systems with symport of weight at most two, but the proof is more difficult.

Theorem 2. *If* $M \in \mathbb{N}OP_2(sym_2)$, *then* $0 \in M \Rightarrow M \in \mathbb{N}FIN$.

Proof. Consider an arbitrary P system Π with two membranes and symport rules of weight at most two, $\Pi = (O, [_1\ [_2\ ]_2\ ]_1, w_1, w_2, E, R_1, R_2, 2)$; without restricting generality we may assume that the objects that compose w_1 and w_2 are disjoint from the objects in E. For Π, consider some computation $\mathcal{C}$ generating 0: $\mathcal{C}$ ends in some configuration C with nothing in membrane 2, $u_1 \in (O-E)^*$ and $u_e \in E^*$ in membrane 1 and $u_0 \in (O-E)^*$ in the environment. Finally, consider an arbitrary halting computation $\mathcal{C}'$ of Π: $\mathcal{C}'$ ends in some configuration C' with $v_2 \in (O-E)^*$ and $v_f \in E^*$ in membrane 2, with $v_1 \in (O-E)^*$ and $v_e \in E^*$ in membrane 1 and $v_0 \in (O-E)^*$ in the environment. We are claiming that $|v_2 v_f| + |v_1 v_e| \leq |w_2| + |w_1|$ (i.e., the total number of objects in the system cannot grow without starting an infinite computation, and thus Π cannot generate numbers greater than the initial number of objects inside it).

Let us assume the contrary. Denote by I_0 the set of objects from $O - E$ that we know must be in the environment in order for Π to halt with region 2 being empty; start with $I_0 = \emptyset$. Since bringing from the environment some object $a \in E \cup I_0$ is necessary (though not sufficient) to increase the number of objects inside the system, some rule $(ab, in) \in R_1$ had to be applied at some step of $\mathcal{C}'$ (if $a \in E$, by definition $b \in O - E$; if $a \in I_0$, then also b must be in $O - E$, otherwise rule (ab, in) would be applicable in C, which is supposed to be a halting configuration).

Clearly, object b was originally in region 1, so it has been brought to the environment by some rule $(b, out) \in R_1$ or $(bc, out) \in R_1$. In the first case, the system cannot halt without "hiding" object b in region 2 (contradiction with the assumption on C). In the second case, we have a few possibilities. If $c = a' \in E$, then by application of rules $(a'b, out)$, (ab, in) we have simply exchanged a' by a in region 1; since a' has been brought in the region 1 beforehand, we can repeat the same reasoning taking a' instead of a (this may only happen a finite number of times since we examine the computation $\mathcal{C}$ backwards). Finally, if $c = b' \in O - E$, then by application of rules $(a'b, out)$, (ab, in) we have exchanged b' by a in region 1. This will not increase the number of objects unless b' does not stay in the environment. Notice also that in configuration C object b' has to be in the environment. Add b' to I_0 and repeat the same reasoning.

The argument above implies that if a system can increase the number of objects inside it, then it cannot halt without any objects in region 2. Therefore, Π cannot generate infinite sets containing 0. □

4 Universality

Theorem 3. $\mathbb{N}OP_2(sym_1, anti_1) = \mathbb{N}_1RE \cup F$, *where* $\mathbb{N}_{\exists 0}SEG_1 \subseteq F \subseteq \mathbb{N}_{\exists 0}FIN$.

Proof. While the upper bound of F results from Theorem 1, the lower bound of F is satisfied even by one-membrane constructions, see [4]. In what follows, we deal with proving $\mathbb{N}_1OP_2(sym_1, anti_1) = \mathbb{N}_1RE$.

Without loss of generality we simulate a counter automaton $M = (d, Q, q_0, q_f, P)$ with the first counter being the *output* counter. Recall that M starts with empty counters. Notice, that the output counter may be only "incremented". We also suppose that all instructions from P are labeled in a one-to-one manner with elements of $\{1, \ldots, n\} = I$, n is a label of the *halt* instruction and $I' = I \backslash \{n\}$. We denote by I_+, I_-, and $I_{=0}$ the set of labels for the "increment", "decrement", and "test for zero" instructions, respectively. We use also the next notation: $C = \{c_k\}, k \in D$.

We construct the P system Π_1 as follows:

$$\Pi_1 = (O, [_1\ [_2\ \ ]_2\ ]_1, w_1, w_2, E, R_1, R_2, 2),$$
$$O = E \cup \{I_c, M, S, T_1, T_2, T_3, J_1, J_2\} \cup \{b_j, d_j \mid j \in I\},$$
$$E = Q \cup C \cup \{a_j, e_j \mid j \in I\} \cup \{a'_j \mid j \in I'\} \cup \{J_0\} \cup \{F_i \mid 0 \leq i \leq 6\},$$

$$w_1 = I_c J_1 J_2,$$
$$w_2 = T_1 T_2 T_3 M S \prod_{j \in I} b_j \prod_{j \in I} d_j,$$
$$R_i = R_{i,s} \cup R_{i,r} \cup R_{i,f}, \quad i = 1, 2.$$

We code the counter automaton as follows:

Region 1 will hold the current state of the automaton, represented by a symbol $q_i \in Q$; region 2 will hold the value of all counters, represented by the number of occurrences of symbols $c_k \in C$, $k \in D$, where $D = \{1, \ldots, d\}$.

We split our proof into several parts that depend on the logical separation of the behavior of the system. We will present the rules and the initial symbols for each part, but we remark that the system we present is the union of all these parts. The rules R_i are given by three phases:

1. START: preparation of the system for the computation.
2. RUN: simulation of instructions of the counter automaton.
3. END: terminating the computation.

The parts of the computations illustrated in the following describe different phases of the evolution of the P system. For simplicity, we focus on explaining a particular phase and omit the objects that do not participate in the evolution at that time. Each rectangle represents a membrane, each variable represents a copy of an object in a corresponding membrane (symbols outside of the outermost rectangle are found in the environment). In each step, the symbols that will evolve (will be moved) are written in **boldface**. The labels of the applied rules are written above the symbol $\Rightarrow$.

1. START

During the first phase we bring from the environment an arbitrary number of symbols c_k, $k \in D$, into region 1. We suppose that we have enough symbols c_k in region 1 to perform the computation. Otherwise, the computation will never stop. We also use the following idea: in our system we have a symbol M which moves from region 2 to region 1 and back in an infinite loop. This loop may be stopped only if all stages completed correctly.

$$R_{1,s} = \{\texttt{1s1} : (I_c, in), \texttt{1s2} : (I_c, out; c_k, in), \texttt{1s3} : (S, out; q_0, in) \mid c_k \in C\}$$
$$R_{2,s} = \{\texttt{2s1} : (M, out), \texttt{2s2} : (M, in), \texttt{2s3} : (S, out; I_c, in)\}$$

Symbol I_c brings symbols c_k from the environment into region 1 (rules `1s1`, `1s2`), where they may be used during the simulation of the "increment" instruction and then moved to region 2.

We illustrate the beginning of the computation as follows:

$$\mathbf{c_{k_1}} c_{k_2} \ldots c_{k_t} q_0 \boxed{\mathbf{I_c} \boxed{\mathbf{M} S}} \Rightarrow^{\texttt{1s2,2s1}} \mathbf{I_c} c_{k_2} \ldots c_{k_t} q_0 \boxed{c_{k_1} \mathbf{M} \boxed{S}} \Rightarrow^{\texttt{1s1,2s2}}$$
$$\mathbf{c_{k_2}} \ldots c_{k_t} q_0 \boxed{\mathbf{I_c} c_{k_1} \boxed{\mathbf{M} S}} \Rightarrow^{\texttt{1s2,2s1}} \ldots \mathbf{I_c} q_0 \boxed{c_{k_1} c_{k_2} \ldots c_{k_t} \mathbf{M} \boxed{S}} \Rightarrow^{\texttt{1s1,2s2}}$$

$$q_0 \boxed{\mathbf{I_c} c_{k_1} c_{k_2} \dots c_{k_t} \boxed{\mathbf{MS}}} \Rightarrow^{\mathtt{2s1,2s3}} \mathbf{q_0} \boxed{\mathbf{S} c_{k_1} c_{k_2} \dots c_{k_t} \mathbf{M} \boxed{I_c}} \Rightarrow^{\mathtt{1s3,2s2}}$$
$$S \boxed{\mathbf{q_0} c_{k_1} c_{k_2} \dots c_{k_t} \boxed{\mathbf{M} I_c}}$$

I_c is eventually exchanged with S, which in turn brings q_0 into region 1, and the simulation of the register machine begins. Symbol I_c is then situated in region 2 and can be used during the second stage as a "trap" symbol, i.e., in order to organize an infinite computation.

Notice that some rules are never executed during a correct simulation: applying them would lead to an infinite computation. To help the reader, we will underline the labels of such rules in the description below.

2. RUN

$$\begin{aligned}
R_{1,r} = {} & \{\mathtt{1r1} : (q_i, out; a_j, in) \mid (j : q_i \to q_l, c_k\gamma) \in P, \gamma \in \{+, -, =0\}\} \\
& \cup \{\mathtt{1r2} : (q_f, out; a_n, in)\} \\
& \cup \{\mathtt{1r3} : (b_j, out; a'_j, in) \mid j \in I'\} \\
& \cup \{\mathtt{1r4} : (a_j, out; J_0, in),\ \mathtt{1r5} : (J_1, out; b_j, in) \mid j \in I\} \\
& \cup \{\mathtt{1r6} : (J_0, out; J_1, in)\} \\
& \cup \{\mathtt{1r7} : (a'_j, out; d_j, in) \mid j \in I_+ \cup I_{=0}\} \\
& \cup \{\mathtt{1r8} : (a'_j, out; a''_j, in),\ \mathtt{1r9} : (a''_j, out; d_j, in) \mid j \in I_-\} \\
& \cup \{\underline{\mathtt{1r10}} : (J_2, out; d_j, in) \mid j \in I_+\} \\
& \cup \{\underline{\mathtt{1r11}} : (J_2, out; J_1, in)\} \\
& \cup \{\mathtt{1r12} : (d_j, out; e_j, in) \mid j \in I\} \\
& \cup \{\mathtt{1r13} : (e_j, out, q_l, in) \mid (j : q_i \to q_l, c_k\gamma) \in P, \gamma \in \{+, -, =0\}\} \\
& \cup \{\mathtt{1r14} : (e_n, out; F_0, in), \mathtt{1r15} : (b_n, out)\}. \\
R_{2,r} = {} & \{\mathtt{2r1} : (b_j, out; a_j, in), \mathtt{2r2} : (a_j, out; J_2, in), \underline{\mathtt{2r3}} : (a_j, out; J_1, in) \mid j \in I\} \\
& \cup \{\underline{\mathtt{2r4}} : (c_k, out; a'_j, in) \mid (j : q_i \to q_l, c_k = 0) \in P\} \\
& \cup \{\mathtt{2r5} : (c_k, out; a''_j, in) \mid (j : q_i \to q_l, c_k -) \in P\} \\
& \cup \{\underline{\mathtt{2r6}} : (a'_j, out; J_1, in) \mid j \in I_{=0} \cup I_+\} \\
& \cup \{\mathtt{2r7} : (a''_j, out) \mid j \in I_-\} \\
& \cup \{\mathtt{2r8} : (a'_j, out; c_k, in) \mid j : q_i \to q_l, c_k +) \in P\} \\
& \cup \{\mathtt{2r9} : (d_j, out; b_j, in) \mid j \in I\} \\
& \cup \{\mathtt{2r10} : (e_j, out; d_j, in),\ \underline{\mathtt{2r11}} : (e_j, out; J_1, in),\ \mathtt{2r12} : (e_j, in) \mid j \in I_+\} \\
& \cup \{\mathtt{2r13} : (J_2, out; d_j, in) \mid j \in I_{=0} \cup I_-\} \\
& \cup \{\mathtt{2r14} : (J_2, out; a'_j, in) \mid j \in I_+\} \\
& \cup \{\underline{\mathtt{2r15}} : (I_c, out; a''_j, in), \mid j \in I_-\} \\
& \cup \{\underline{\mathtt{2r16}} : (I_c, out; J_0, in)\}
\end{aligned}$$

First of all, we mention that if during the phase RUN object J_2 comes to the environment (**Scenario 0**), it remains there forever. Then during the simulation of the next instruction of the counter automaton, instead of rule `2r2`, rule `2r3` will be applied, sending J_1 forever to region 2. Then the computation never halts, see scenario 1 below.

Let us explain the synchronization of a_j coming to the environment and b_j leaving the environment: the first one brings J_0 into region 1 while the latter brings J_1 into the environment; then rule `1r6` returns J_0 and J_1 to their original locations.

If a_j comes to the environment without b_j leaving it, J_1 remains in region 1 (or 2) and J_0 comes to region 1 (**Scenario 1**), so `2r16` is applied, causing an endless computation since `1s1` and `1s2` are always applicable.

If b_j leaves the environment without a_j coming there, J_0 remains in the environment and J_1 comes there (**Scenario 2**), so `1r11` is applied (immediately in case of simulating an increment instruction or in a few steps in case of simulating a decrement or a zero-test instruction), sending J_2 forever to the environment. The computation never halts, see scenario 0.

We also mention that applying rule `1r10` causes scenario 0; applying `2r4` leads to applying `2r6`; applying rule `2r6` or `2r11` causes J_1 to stay in region 2 forever, eventually causing scenario 1. Finally, applying rule `2r15` also causes an infinite computation by `1s1` and `1s2`. Therefore, in order for a computation to halt, no underlined rule should be applied.

We will now consider the "main" line of computation.

"Increment" instruction:
(i) *There is some c_k in region 1:*

$$q_l e_j \mathbf{a_j} a'_j J_0 \boxed{\mathbf{q_i} c_k J_1 J_2 \boxed{b_j d_j}} \Rightarrow^{1r1} q_i q_l e_j a'_j J_0 \boxed{\mathbf{a_j} c_k J_1 J_2 \boxed{\mathbf{b_j} d_j}} \Rightarrow^{2r1}$$

$$q_i q_l e_j \mathbf{a'_j} J_0 \boxed{\mathbf{b_j} c_k J_1 \mathbf{J_2} \boxed{\mathbf{a_j} d_j}} \Rightarrow^{1r3,2r2} q_i q_l e_j \mathbf{b_j J_0} \boxed{\mathbf{a'_j} c_k \mathbf{J_1 a_j} \boxed{\mathbf{J_2} d_j}} \Rightarrow^{1r4,1r5,2r14}$$

$$q_i q_l e_j a_j \mathbf{J_1} \boxed{\mathbf{c_k J_0 b_j} J_2 \boxed{\mathbf{a'_j d_j}}} \qquad \text{(A)}$$

$$\Rightarrow^{1r6,2r8,2r9} q_i q_l \mathbf{e_j} a_j J_0 \boxed{a'_j J_1 \mathbf{d_j} J_2 \boxed{c_k b_j}} \Rightarrow^{1r12}$$

$$q_i q_l \mathbf{d_j} a_j J_0 \boxed{\mathbf{a'_j} J_1 \mathbf{e_j} J_2 \boxed{c_k b_j}} \Rightarrow^{1r7,2r12} q_i q_l a_j a'_j J_0 \boxed{\mathbf{d_j} J_1 J_2 \boxed{\mathbf{e_j} c_k b_j}} \Rightarrow^{2r10}$$

$$q_i \mathbf{q_l} a_j a'_j J_0 \boxed{\mathbf{e_j} J_1 J_2 \boxed{d_j c_k b_j}} \Rightarrow^{1r13} q_i e_j a_j a'_j J_0 \boxed{\mathbf{q_l} J_1 J_2 \boxed{d_j c_k b_j}}$$

In that way, q_i is replaced by q_l and c_k is moved from region 1 into region 2. Notice that symbols a_j, b_j, a'_j, d_j, e_j, J_2, J_1, J_0 have returned to their original positions.

(ii) *There is no c_k in region 1:*
Consider configuration (A) above without object c_k in region 1:

$$q_i q_l e_j a_j \mathbf{J_1} \boxed{\mathbf{J_0 b_j} J_2 \boxed{a'_j \mathbf{d_j}}} \Rightarrow^{1r6,2r9} q_i q_l e_j a_j J_0 \boxed{\mathbf{J_1 d_j} J_2 \boxed{\mathbf{a'_j} b_j}}$$

Now rule 2r6 will be applied, causing an infinite computation.

"Decrement" instruction:

(i) *There is some c_k in region 2:*

$$q_l e_j a_t \mathbf{a_j} a'_j a''_j J_0 \boxed{\mathbf{q_i} J_1 J_2 \boxed{b_j c_k d_j}} \Rightarrow^{1r1} q_i q_l e_j a_t a'_j a''_j J_0 \boxed{\mathbf{a_j} J_1 J_2 \boxed{\mathbf{b_j} c_k d_j}} \Rightarrow^{2r1}$$

$$q_i q_l e_j a_t \mathbf{a'_j} a''_j J_0 \boxed{\mathbf{b_j} J_1 \mathbf{J_2} \boxed{\mathbf{a_j} c_k d_j}} \Rightarrow^{1r3,2r2} q_i q_l e_j a_t \mathbf{a''_j b_j J_0} \boxed{\mathbf{a'_j J_1 a_j} \boxed{J_2 c_k d_j}}$$

$$\Rightarrow^{1r4,1r5,1r8} q_i q_l e_j a_t a_j a'_j \mathbf{J_1} \boxed{\mathbf{J_0 b_j a''_j} \boxed{J_2 \mathbf{c_k d_j}}} \qquad \text{(B)}$$

$$\Rightarrow^{1r6,2r5,2r9} q_i q_l \mathbf{e_j} a_t a_j a'_j J_0 \boxed{J_1 \mathbf{d_j} c_k \boxed{\mathbf{a''_j} b_j J_2}} \Rightarrow^{1r12,2r7}$$

$$q_i \mathbf{q_l d_j} a_t a_j a'_j J_0 \boxed{\mathbf{a''_j} J_1 \mathbf{e_j} c_k \boxed{b_j J_2}} \Rightarrow^{1r9,1r13} q_i e_j \mathbf{a_t} a_j a'_j a''_j J_0 \boxed{\mathbf{q_l} J_1 \mathbf{d_j} c_k \boxed{b_j \mathbf{J_2}}}$$

$$\Rightarrow^{1r1,2r13} q_i q_l e_j a_j a'_j a''_j J_0 \boxed{\mathbf{a_t} J_1 J_2 c_k \boxed{b_j d_j}}$$

In the way described above, q_i is replaced by q_l and c_k is removed from region 2 to region 1. Notice that symbols a_j, a'_j, a''_j, b_j, d_j, e_j, J_2, J_1, J_0 have returned to their original positions. Symbol d_j returns to region 2 in the first step of the simulation of the next instruction (the last step of the illustration).

(ii) *There is no c_k in region 2:*

We start with configuration (B) without c_k in region 2.

$$q_i q_l e_j a_t a_j a'_j \mathbf{J_1} \boxed{\mathbf{J_0 b_j a''_j} \boxed{J_2 \mathbf{d_j I_c}}}$$

Now rule 2r15 will be applied, leading to an infinite computation.

"Test for zero" instruction:

q_i is replaced by q_l if there is no c_k in region 2, otherwise a'_j in region 1 exchanges with c_k in region 2 and the computation will never stop.

(i) *There is no c_k in region 2:*

$$q_l e_j a_t \mathbf{a_j} a'_j J_0 \boxed{\mathbf{q_i} J_1 J_2 \boxed{b_j d_j}} \Rightarrow^{1r1} q_i q_l e_j a_t a'_j J_0 \boxed{\mathbf{a_j} J_1 J_2 \boxed{\mathbf{b_j} d_j}}$$

$$\Rightarrow^{2r1} q_i q_l e_j a_t \mathbf{a'_j} J_0 \boxed{\mathbf{b_j} J_1 \mathbf{J_2} \boxed{\mathbf{a_j} d_j}}$$

$$\Rightarrow^{1r3,2r2} q_i q_l e_j a_t \mathbf{b_j J_0} \boxed{a'_j \mathbf{J_1 a_j} \boxed{J_2 d_j}} \qquad \text{(C)}$$

$$\Rightarrow^{1r4,1r5} q_i q_l e_j a_t a_j \mathbf{J_1} \boxed{\mathbf{b_j} a'_j \mathbf{J_0} \boxed{J_2 \mathbf{d_j}}} \Rightarrow^{1r6,2r9} q_i q_l \mathbf{e_j} a_t a_j J_0 \boxed{J_1 a'_j \mathbf{d_j} \boxed{J_2 b_j}} \Rightarrow^{1r12}$$

$$q_i \mathbf{q_l} a_t a_j \mathbf{d_j} J_0 \boxed{J_1 \mathbf{a'_j e_j} \boxed{J_2 b_j}} \Rightarrow^{1r7,1r13} q_i e_j \mathbf{a_t} a_j a'_j J_0 \boxed{J_1 \mathbf{q_l d_j} \boxed{\mathbf{J_2} b_j}} \Rightarrow^{1r1,2r13}$$

$$q_i q_l e_j a_j a'_j J_0 \boxed{\mathbf{a_t} J_1 J_2 \boxed{b_j d_j}}$$

In this case, q_i is replaced by q_l. Notice that symbols a_j, a'_j, b_j, d_j, e_j, J_2, J_1, J_0 have returned to their original positions. Symbol d_j returns to region 2 in the first step of the simulation of the next instruction (the last step of the illustration).

(ii) *There is some c_k in region 2:*
Consider configuration (C) with object c_k in region 2:

$$q_i q_l e_j a_t \mathbf{b_j J_0} \boxed{\mathbf{a'_j J_1 a_j} \boxed{J_2 \mathbf{c_k} d_j}}$$

Now rule 2r4 will be applied immediately, and then the computation never halts.

Let us consider the symbols from region 2 visiting the environment and going back: $2 \to 1 \to 0 \to 1 \to 2$ (b_j, d_j for all instructions) and the symbols from the environment visiting region 2 and going back: $0 \to 1 \to 2 \to 1 \to 0$. The latter ones are: a_j for all instructions, and also $\{a'_j, e_j \mid j \in I_+\} \cup \{a''_j \mid j \in I_-\}$.

Then we have to argue that if they **Return** to their "home region" ($2 \to 1 \to 2$ or $0 \to 1 \to 0$) or **Repeat** their visit to the "opposite region" before returning "home" ($2 \to 1 \to 0 \to 1 \to 0$ or $0 \to 1 \to 2 \to 1 \to 2$), an infinite computation is unavoidable, or such case is not possible.

$a_j, j \in I$. Return: see scenario 1; repeat: impossible without b_j.

$a'_j, j \in I_+$. Return: impossible without d_j; repeat: impossible without J_2.

$e_j, j \in I_+$. Return: since d_j does not come to region 2, see repeat for d_j, below; repeat: 2r11.

$a''_j, j \in I_-$. Return: impossible without d_j; repeat: in the same step q_l is brought into region 1, which will require J_2 in region 1 in three steps. Since d_j stays in region 2 for at least two steps, J_2 is unavailable in region 1 for at least three steps, so 2r3 is applied.

$b_j, j \in I$. Return: if a_j comes to the environment, scenario 1 takes place. If a_j returns to region 2, rule 2r3 is applied.

$d_j, j \in I_+$. Return: impossible without e_j; repeat: 1r10.

$d_j, j \in I_- \cup I_{=0}$. Return: e_j stays in the environment, the simulation stops and the computation never ends due to 2s1, 2s2; repeat: in the same step a_k is brought into region 2, which will require J_2 in region 1 in two steps. Since J_2 is unavailable in region 1 for at least two steps, 2r3 is applied.

3. END

$$\begin{aligned}
R_{1,f} = {} & \{\texttt{1f1} : (T_1, out; F_1, in)\} \cup \{\texttt{1f2} : (F_i, out; F_{i+1}, in) \mid 1 \le i \le 5\} \\
& \cup \{\texttt{1f3} : (T_2, out),\ \texttt{1f4} : (M, out; T_2, in),\ \texttt{1f5} : (J_2, out; T_2, in)\} \\
& \cup \{\texttt{1f6} : (T_3, out), \texttt{1f7} : (b_j, out; T_3, in), \texttt{1f8} : (d_j, out; T_3, in) \mid j \in I\}. \\
R_{2,f} = {} & \{\texttt{2f1} : (F_0, out), \texttt{2f2} : (J_2, out; F_0, in), \texttt{2f3} : (T_i, out; F_0, in) \mid 1 \le i \le 3\} \\
& \cup \{\texttt{2f4} : (F_6, out), \texttt{2f5} : (b_j, out; F_6, in), \texttt{2f6} : (d_j, out; F_6, in), \\
& \quad \texttt{2f7} : (a_j, out; F_6, in) \mid j \in I\} \cup \{\texttt{2f8} : (e_j, out; F_6, in) \mid j \in I_+\}.
\end{aligned}$$

Once the register machine reaches the final state, q_f is in region 1 and it exchanges with object a_n (rule 1r2). If on the previous steps of simulation of

counter automaton M object J_1 was moved to region 2 (by rules 2r6, 2r11), rule 2r16 will be applied, and the computation never halts. If during the previous steps of simulation of counter automaton M object J_2 was moved to the environment (by rules 1r10, 1r11), rule 2r3 will be applied, leading to an infinite computation. If not, i.e., if object J_2 is present in region 1, it will be moved to region 2 (rule 2r2), then object F_0 will be moved to region 1 in several steps (rules 1r14).

It takes T_1, T_2, T_3 and J_2 to region 1, in either order. The duty of T_2 is to bring J_2 and M to the environment (J_2 can be brought to the environment immediately, or after M if the latter immediately goes to the environment; the object M can oscillate for indefinite time, but we are interested in halting computations). The duty of T_3 is to bring b_j and d_j to the environment. T_1 starts a chain of exchanges of objects F_i, as a result object F_6 will be moved to region 1 and then it removes objects b_j, d_j (and possible objects a_j, e_j) from region 2.

We illustrate the end of computations as follows:

$$F_1F_2F_3F_4F_5F_6\left[\mathbf{F_0}\left[\mathbf{MT_1}T_2T_3J_2b_jd_j\right]\right] \Rightarrow^{2f3,2s1}$$

$$\mathbf{F_1}F_2F_3F_4F_5F_6\left[\mathbf{T_1M}\left[\mathbf{F_0}T_2T_3J_2b_jd_j\right]\right] \Rightarrow^{1f1,2f1,2s2}$$

$$\mathbf{F_2}F_3F_4F_5F_6T_1\left[\mathbf{F_1F_0}\left[\mathbf{M}T_2\mathbf{T_3}J_2b_jd_j\right]\right] \Rightarrow^{1f2,2f3,2s1}$$

$$F_1\mathbf{F_3}F_4F_5F_6T_1\left[\mathbf{F_2T_3M}\left[\mathbf{F_0}T_2J_2b_jd_j\right]\right] \Rightarrow^{1f2,2f1,2s2}$$

$$F_1F_2\mathbf{F_4}F_5F_6T_1T_3\left[\mathbf{F_3F_0}\left[\mathbf{M}T_2\mathbf{J_2}b_jd_j\right]\right] \Rightarrow^{1f2,2f2,2s1}$$

$$F_1F_2F_3\mathbf{F_5}F_6T_1T_3\left[\mathbf{F_4}J_2\mathbf{M}\left[\mathbf{F_0}T_2b_jd_j\right]\right] \Rightarrow^{1f2,2f1,2s2}$$

$$F_1F_2F_3F_4\mathbf{F_6}T_1T_3\left[\mathbf{F_5}J_2\mathbf{F_0}\left[\mathbf{MT_2}b_jd_j\right]\right] \Rightarrow^{1f2,2f3,2s1}$$

$$F_1\ldots F_5T_1T_3\left[\mathbf{F_6}J_2\mathbf{T_2M}\left[\mathbf{F_0b_j}d_j\right]\right] \Rightarrow^{1f3,2f1,2f5,2s2}$$

$$F_1\ldots F_5T_1\mathbf{T_2T_3}\left[\mathbf{J_2b_j}F_0\left[\mathbf{MF_6}d_j\right]\right] \Rightarrow^{1f5,1f7,2f4,2s1}$$

$$F_1\ldots F_5b_jJ_2T_1\left[\mathbf{T_2T_3}F_0\mathbf{MF_6}\left[\mathbf{d_j}\right]\right] \Rightarrow^{1f3,1f6,2f6,2s2}$$

$$F_1\ldots F_5b_jJ_2T_1T_2\mathbf{T_3}\left[\mathbf{d_j}F_0\left[\mathbf{MF_6}\right]\right] \Rightarrow^{1f8,2f4,2s1}$$

$$F_1\ldots F_5b_jd_jJ_2T_1\mathbf{T_2}\left[\mathbf{T_3}F_0\mathbf{M}F_6\left[\right]\right] \Rightarrow^{1f4,1f6}$$

$$F_1\ldots F_5b_jd_jJ_2MT_1T_3\left[\mathbf{T_2}F_0F_6\left[\right]\right] \Rightarrow^{1f3}$$

$$F_1\ldots F_5b_jd_jJ_2MT_1T_2T_3\left[F_0F_6\left[\right]\right]$$

We continue in this manner until all objects b_j, d_j (and possible objects a_j, e_j) from the elementary membrane 2 have been moved to the environment. Notice that the result in the elementary membrane 2 (multiset c_1^t) cannot be changed

during phase END, as object J_2 now is situated in the environment. Thus, object a'_j cannot enter into region 2 by rule `2r14` and therefore cannot bring object c_k into region 2 by rule `2r8`. Recall that the counter automaton can only *increment* the first counter c_1, so all other computations of P system Π_1 cannot change the number of symbols c_1 in the elementary membrane. Thus, at the end of a terminating computation, in the elementary membrane there are the result (multiset c_1^t) and only the one additional object I_c. □

Theorem 4. $\mathbb{N}OP_2(sym_2) = \mathbb{N}_1RE \cup F$, *where* $\mathbb{N}_{\ni 0}SEG_1 \cup \mathbb{N}_{\ni 0}SEG_2 \subseteq F \subseteq \mathbb{N}_{\ni 0}FIN$.

Proof. While the upper bound of F results from Theorem 2, the lower bound of F is satisfied even by one-membrane constructions, see [4]. In what follows, we deal with proving $\mathbb{N}_1OP_2(sym_2) = \mathbb{N}_1RE$.

As in the proof of Theorem 3 we simulate a counter automaton $M = (d, Q, q_0, q_f, P)$ that starts with empty counters and we suggest that the output counter may be only "incremented". Again we suppose that all instructions from P are labeled in a one-to-one manner with elements of $\{1, \ldots, n\} = I$, n is a label of the *halt* instruction, $I' = I \setminus \{n\}$, and $I = I_+ \cup I_- \cup I_{=0}$, where we denote by I_+, I_-, and $I_{=0}$ the set of labels for the "increment", "decrement", and "test for zero" instructions, respectively. We use also the next notations: $C = \{c_i \mid 1 \leq i \leq d\}$, $Q' = Q \setminus \{q_0\}$, and $\hat{Q} = \{\hat{q}_i,\ 0 \leq i \leq f-1\}$.

We construct the P system Π_2 as follows:

$$\Pi_2 = (O, [_1\ [_2\ ]_2\ ]_1, w_1, w_2, E, R_1, R_2, 2),$$
$$O = E \cup \hat{Q} \cup \{\breve{a}_j, b_j, d_j, f_j \mid j \in I'\} \cup \{\$_1, \$_2, \$_3, \breve{b}, I_c, t_1, t_2, t_3, t_5, t_7, t_9\},$$
$$E = C \cup Q' \cup \{a_j, \breve{b}_j, e_j \mid j \in I'\} \cup \{\#, F, t_4, t_6, t_8\},$$
$$w_1 = q_0 I_c \breve{b} \$_1 \$_3 t_1 t_5 t_7 t_9 \prod_{j \in I'} \breve{a}_j \prod_{j \in I'} f_j \prod_{0 \leq i \leq f-1} \hat{q}_i$$
$$w_2 = \$_2 t_2 t_3 \prod_{j \in I'} b_j \prod_{j \in I'} d_j,$$
$$R_i = R_{i,s} \cup R_{i,r} \cup R_{i,f}, i \in \{1, 2\}.$$

We code the counter automaton as in Theorem 3 above: region 1 will hold the current state of the automaton, represented by a symbol $q_i \in Q$; region 2 will hold the value of all counters, represented by the number of occurrences of symbols $c_k \in C$, $k \in D$, where $D = \{1, \cdots, d\}$. We also use the following idea (called "*Circle*") realized by phase "START" below: from the environment, we bring symbols c_k into region 1 all the time during the computation. This process may only be stopped if all phases finish correctly; otherwise, the computation will never stop.

The rules R_i are given by three phases:

1. START: preparation of the system for the computation.
2. RUN: simulation of instructions of the counter automaton.
3. END: terminating the computation.

As in Theorem 3 we split our proof into several parts that depend on the logical separation of the behavior of the system and use the same agreements.

1. START

$$R_{1,s} = \{\texttt{1s1} : (I_c, out), \texttt{1s2} : (I_c c_k, in) \mid k \in D\},$$
$$R_{2,s} = \emptyset.$$

Symbol I_c brings one symbol $c \in C$ from the environment into region 1 (rules `1s1, 1s2`) where it may be used during the simulation of an "increment" instruction and moved to region 2.

2. RUN

$$\begin{aligned}
R_{1,r} = {} & \{\texttt{1r1} : (q_i \hat{q}_i, out) \mid 0 \le i \le f-1\} \\
& \cup \{\texttt{1r2} : (a_j \hat{q}_i, in) \mid (j : q_i \to q_l, k\gamma) \in P, \gamma \in \{+, -, =0\}, k \in D\} \\
& \cup \{\texttt{1r3} : (a_j b_j, out),\ \texttt{1r4} : (b_j \check{b}_j, in) \mid j \in I'\} \\
& \cup \{\texttt{1r5} : (\$_1\$_2, out),\ \texttt{1r6} : (\#\$_2, in)\} \\
& \cup \{\texttt{1r7} : (\check{b}_j d_j, out) \mid j \in I_+ \cup I_-\} \\
& \cup \{\texttt{1r8} : (d_j q_l, in) \mid (j : q_i \to q_l, k-) \in P, k \in D\} \\
& \cup \{\texttt{1r9} : (\check{a}_j \check{b}_j, out) \mid j \in I_{=0}\} \\
& \cup \{\texttt{1r10} : (\check{a}_j q_l, in) \mid (j : q_i \to q_l, k = 0) \in P, k \in D\} \\
& \cup \{\texttt{1r11} : (d_j e_j, in),\ \texttt{1r12} : (e_j f_j, out), \\
& \quad \texttt{1r13} : (f_j q_l, in) \mid (j : q_i \to q_l, k+) \in P, k \in D\}. \\
R_{2,r} = {} & \{\texttt{2r1} : (a_j \check{a}_j, in),\ \texttt{2r2} : (a_j b_j, out) \mid j \in I'\} \\
& \cup \{\texttt{2r3} : (\check{a}_j c_k, out) \mid (j : q_i \to q_l, k-) \in P, k \in D\} \\
& \cup \{\texttt{2r4} : (\check{a}_j \$_2, out),\ \texttt{2r5} : (a_j \$_2, out) \mid j \in I_-\} \\
& \cup \{\texttt{2r6} : (\check{a}_j, out) \mid j \in I_+\} \\
& \cup \{\texttt{2r7} : (\#, in),\ \texttt{2r8} : (\#, out)\} \\
& \cup \{\texttt{2r9} : (b_j \check{b}_j, in) \mid j \in I_- \cup I_{=0}\} \\
& \cup \{\texttt{2r10} : (b_j \check{b}, in) \mid j \in I_+\} \\
& \cup \{\texttt{2r11} : (\check{b}_j d_j, out) \mid j \in I_+ \cup I_-\} \\
& \cup \{\texttt{2r12} : (d_j \check{b}, in) \mid j \in I_-\} \\
& \cup \{\texttt{2r13} : (\check{b}, out)\} \\
& \cup \{\texttt{2r14} : (\check{b}_j c_k, in) \mid (j : q_i \to q_l, k+) \in P, k \in D\} \\
& \cup \{\texttt{2r15} : (d_j e_j, in),\ \texttt{2r16} : (d_j \$_3, in),\ \texttt{2r17} : (e_j, out), \\
& \quad \texttt{2r18} : (\check{b}_j \$_3, in) \mid j \in I_+\} \\
& \cup \{\texttt{2r19} : (\$_2\$_3, out)\} \\
& \cup \{\texttt{2r20} : (\check{a}_j c_k, out) \mid (j : q_i \to q_l, k = 0) \in P, k \in D\} \\
& \cup \{\texttt{2r21} : (b_j \$_3, in),\ \texttt{2r22} : (\check{a}_j \check{b}_j, out),\ \texttt{2r23} : (\check{b}_j \$_2, out) \mid j \in I_{=0}\}.
\end{aligned}$$

"Increment" instruction:

$$q_l a_j e_j \# \boxed{\mathbf{q_i}\hat{\mathbf{q}}_\mathbf{i}\check{a}_j f_j \check{b}\$_1\$_3 \boxed{b_j d_j \$_2}} \Rightarrow^{\texttt{1r1}} q_l q_i \hat{\mathbf{q}}_\mathbf{i}\mathbf{a_j} e_j \# \boxed{\check{a}_j f_j \check{b}\$_1\$_3 \boxed{b_j d_j \$_2}} \Rightarrow^{\texttt{1r2}}$$

$$q_l q_i e_j \# \boxed{\hat{q}_i \mathbf{a_j}\check{\mathbf{a}}_\mathbf{j} f_j \check{b}\$_1\$_3 \boxed{b_j d_j \$_2}} \Rightarrow^{\texttt{2r1}} q_l q_i e_j \# \boxed{\hat{q}_i f_j \check{b}\$_1\$_3 \boxed{\mathbf{b_j a_j}\check{\mathbf{a}}_\mathbf{j} d_j \$_2}} \Rightarrow^{\texttt{2r2,2r6}}$$

$$q_l q_i e_j \# \boxed{\hat{q}_i b_j a_j \check{a}_j f_j \check{b}\$_1\$_3 \boxed{d_j \$_2}} \qquad \text{(A)}$$

Now there are two variants of computations (depending on the application of rule `2r1` or rule `1r3`). Consider applying rule `2r1`:

$$q_l q_i \check{b}_j e_j \# \boxed{\hat{q}_i \mathbf{b_j a_j}\check{\mathbf{a}}_\mathbf{j} f_j \check{\mathbf{b}}\$_1\$_3 \boxed{d_j \$_2}} \Rightarrow^{\texttt{2r1,2r10}} q_l q_i \check{b}_j e_j \# \boxed{\hat{q}_i f_j \$_1\$_3 \boxed{\mathbf{b_j a_j}\check{\mathbf{a}}_\mathbf{j}\check{\mathbf{b}} d_j \$_2}}$$

$$\Rightarrow^{\texttt{2r2,2r6,2r13}} q_l q_i \check{b}_j e_j \# \boxed{\hat{q}_i b_j a_j \check{a}_j f_j \check{b}\$_1\$_3 \boxed{d_j \$_2}}$$

Thus, we come back to configuration (A) above. As we are interested only in finite computations, we assume that rule `1r3` will be eventually applied:

$$q_l q_i \check{b}_j e_j \# \boxed{\hat{q}_i \mathbf{b_j a_j}\check{a}_j f_j \check{b}\$_1\$_3 \boxed{d_j \$_2}} \Rightarrow^{\texttt{1r3}} q_l q_i a_j \mathbf{b_j}\check{\mathbf{b}}_\mathbf{j} e_j \# \boxed{\hat{q}_i \check{a}_j f_j \check{b}\$_1\$_3 \boxed{d_j \$_2}}$$

$$\Rightarrow^{\texttt{1r4}} q_l q_i a_j e_j \# \boxed{\hat{q}_i \check{a}_j f_j \check{b} b_j \check{b}_j \$_1\$_3 \boxed{d_j \$_2}}$$

Now there are two cases: object c_k is present or is not present in region 1. The last case leads to an infinite computation. Indeed, object $\$_3$ will be moved to region 2 by rule `2r18`, and it comes back to region 1 with object $\$_2$ by rule `2r19`. Notice, that the case then object $\$_2$ appears in region 1, eventually leads to an infinite computation by rules `1r5`, `1r6` and `2r7`, `2r8`. Consider the first case, i.e. then object c_k is present in region 1:

$$q_l q_i a_j e_j \# \boxed{\hat{q}_i \check{a}_j f_j \check{\mathbf{b}}\mathbf{b_j}\check{\mathbf{b}}_\mathbf{j}\mathbf{c_k}\$_1\$_3 \boxed{d_j \$_2}} \Rightarrow^{\texttt{2r10,2r14}}$$

$$q_l q_i a_j e_j \# \boxed{\hat{q}_i \check{a}_j f_j \$_1\$_3 \boxed{\check{\mathbf{b}} b_j \check{\mathbf{b}}_\mathbf{j}\mathbf{d_j} c_k \$_2}} \Rightarrow^{\texttt{2r11,2r13}}$$

$$q_l q_i a_j e_j \# \boxed{\hat{q}_i \check{a}_j f_j \check{b}_j d_j \check{b}\$_1\$_3 \boxed{b_j c_k \$_2}}$$

Notice, that now rule `2r14` cannot be applied again, as in this case rule `2r16` will be applied, which leads to an infinite computation (by rules `2r19`, `1r5`, `1r6` and `2r7`, `2r8`). Otherwise rule `1r7` will be applied:

$$q_l q_i a_j e_j \# \boxed{\hat{q}_i \check{a}_j f_j \check{\mathbf{b}}_\mathbf{j}\mathbf{d_j}\check{b}\$_1\$_3 \boxed{b_j c_k \$_2}} \Rightarrow^{\texttt{1r7}} q_l q_i a_j \check{b}_j \mathbf{d_j e_j} \# \boxed{\hat{q}_i \check{a}_j f_j \check{b}\$_1\$_3 \boxed{b_j c_k \$_2}}$$

$$\Rightarrow^{\texttt{1r11}} q_l q_i a_j \check{b}_j \# \boxed{\hat{q}_i \check{a}_j d_j e_j f_j \check{b}\$_1\$_3 \boxed{b_j c_k \$_2}}$$

Notice, that rule `1r12` cannot be applied, as in this case applying of rule `2r16` leads to an infinite computation (see above). So, rule `2r15` will be applied:

$$q_l q_i a_j \check{b}_j \# \boxed{\hat{q}_i \breve{a}_j \mathbf{d_j e_j} f_j \check{b} \$_1 \$_3 \boxed{b_j c_k \$_2}} \Rightarrow^{\texttt{2r15}} q_l q_i a_j \check{b}_j \# \boxed{\hat{q}_i \breve{a}_j f_j \check{b} \$_1 \$_3 \boxed{b_j c_k d_j \mathbf{e_j} \$_2}}$$

$$\Rightarrow^{\texttt{2r17}} q_l q_i a_j \check{b}_j \# \boxed{\hat{q}_i \breve{a}_j \mathbf{e_j f_j} \check{b} \$_1 \$_3 \boxed{b_j c_k d_j \$_2}} \Rightarrow^{\texttt{1r12}}$$

$$\mathbf{f_j q_l} q_i a_j \check{b}_j e_j \# \boxed{\hat{q}_i \breve{a}_j \check{b} \$_1 \$_3 \boxed{b_j c_k d_j \$_2}} \Rightarrow^{\texttt{1r13}} q_i a_j \check{b}_j e_j \# \boxed{q_l \hat{q}_i \breve{a}_j \check{b} f_j \$_1 \$_3 \boxed{b_j c_k d_j \$_2}}$$

In that way, q_i is replaced by q_l and c_k is moved from region 1 into region 2. Notice that symbols $a_j, b_j, d_j, e_j, f_j, \check{b}_j, \hat{q}_i, \check{b}$ have returned to their original positions.

"Decrement" instruction:

$$q_l a_j \check{b}_j \# \boxed{\mathbf{q_i \hat{q}_i} \breve{a}_j \check{b} \$_1 \$_3 \boxed{b_j d_j \$_2}} \Rightarrow^{\texttt{1r1}} q_l q_i \mathbf{\hat{q}_i a_j} \check{b}_j \# \boxed{\breve{a}_j \check{b} \$_1 \$_3 \boxed{b_j d_j \$_2}} \Rightarrow^{\texttt{1r2}}$$

$$q_l q_i \check{b}_j \# \boxed{\hat{q}_i \mathbf{a_j \breve{a}_j} \check{b} \$_1 \$_3 \boxed{b_j d_j \$_2}} \Rightarrow^{\texttt{2r1}} q_l q_i \check{b}_j \# \boxed{\hat{q}_i \check{b} \$_1 \$_3 \boxed{b_j a_j \breve{a}_j d_j \$_2}}$$

Now there are two cases: object c_k is present or is not present in region 2. The last case leads to an infinite computation, as object $\$_2$ appears in region 1 by rule `2r4` (see above). If object c_k is present in region 2, then rule `2r3` will be applied:

$$q_l q_i \check{b}_j \# \boxed{\hat{q}_i \check{b} \$_1 \$_3 \boxed{\mathbf{b_j a_j \breve{a}_j c_k} d_j \$_2}} \Rightarrow^{\texttt{2r2,2r3}} q_l q_i \check{b}_j \# \boxed{b_j a_j \breve{a}_j c_k \hat{q}_i \check{b} \$_1 \$_3 \boxed{d_j \$_2}}$$

Now there are two possibilities: rule `1r3` or `2r1` may be applied. The last case leads to an infinite computation, as object $\$_2$ will be moved to region 1 by rule `2r5` (rule `2r2` cannot be applied, as now object b_j is situated in region 1). So, consider applying rule `1r3`:

$$q_l q_i \check{b}_j \# \boxed{\mathbf{b_j a_j} \breve{a}_j c_k \hat{q}_i \check{b} \$_1 \$_3 \boxed{d_j \$_2}} \Rightarrow^{\texttt{1r3}} q_l q_i a_j \mathbf{b_j \check{b}_j} \# \boxed{\breve{a}_j c_k \hat{q}_i \check{b} \$_1 \$_3 \boxed{d_j \$_2}} \Rightarrow^{\texttt{1r4}}$$

$$q_l q_i a_j \# \boxed{\breve{a}_j c_k \hat{q}_i \check{b} \mathbf{b_j \check{b}_j} \$_1 \$_3 \boxed{d_j \$_2}} \Rightarrow^{\texttt{2r9}} q_l q_i a_j \# \boxed{\breve{a}_j c_k \hat{q}_i \check{b} \$_1 \$_3 \boxed{b_j \mathbf{\check{b}_j d_j} \$_2}} \Rightarrow^{\texttt{2r11}}$$

$$q_l q_i a_j \# \boxed{\breve{a}_j c_k \hat{q}_i \check{b}_j d_j \check{b} \$_1 \$_3 \boxed{b_j \$_2}}$$

Now there are two possibilities: rule `1r7` or `2r12` may be applied. The last case leads to an infinite computation by "Circle", i.e. by rules `1s1`, `1s2`. So, consider applying rule `1r7`:

$$q_l q_i a_j a_t \# \boxed{\breve{a}_j \breve{a}_t c_k \hat{q}_i \hat{q}_l \mathbf{\check{b}_j d_j} \check{b} \$_1 \$_3 \boxed{b_j \$_2}} \Rightarrow^{\texttt{1r7}} \mathbf{d_j q_l} q_i a_j a_t \check{b}_j \# \boxed{\breve{a}_j \breve{a}_t c_k \hat{q}_i \hat{q}_l \check{b} \$_1 \$_3 \boxed{b_j \$_2}}$$

$$\Rightarrow^{\texttt{1r8}} q_i a_j a_t \check{b}_j \# \boxed{\breve{a}_j \breve{a}_t c_k \hat{q}_i \mathbf{\hat{q}_l q_l d_j \check{b}} \$_1 \$_3 \boxed{b_j \$_2}}$$

$$\Rightarrow^{\texttt{1r1,2r12}} q_i q_l a_j \mathbf{\hat{q}_l a_t} \check{b}_j \# \boxed{\breve{a}_j \breve{a}_t c_k \hat{q}_i \$_1 \$_3 \boxed{b_j d_j \mathbf{\check{b}} \$_2}}$$

In the way described above, q_i is replaced by q_l and c_k is removed from region 2 to region 1. Notice that symbols $a_j, b_j, d_j, \check{b}_j, \hat{q}_i, \check{b}$ have returned to their original

positions. Symbol d_j returns to region 2 in the first step of the simulation of the next instruction (the last step of the illustration) and symbol $\check{b}$ in the second step of the simulation of the next instruction.

"Test for zero" instruction:

$$q_l a_j \check{b}_j \# \boxed{\mathbf{q_i}\hat{\mathbf{q}}_\mathbf{i}\breve{a}_j\$_1\$_3\boxed{b_j\$_2}} \Rightarrow^{\texttt{1r1}} q_l q_i \hat{\mathbf{q}}_\mathbf{i}\mathbf{a_j}\check{b}_j \# \boxed{\breve{a}_j\$_1\$_3\boxed{b_j\$_2}} \Rightarrow^{\texttt{1r2}}$$

$$q_l q_i \check{b}_j \# \boxed{\hat{q}_i\mathbf{a_j}\breve{\mathbf{a}}_\mathbf{j}\$_1\$_3\boxed{b_j\$_2}} \Rightarrow^{\texttt{2r1}} q_l q_i \check{b}_j \# \boxed{\hat{q}_i\$_1\$_3\boxed{b_j a_j \breve{a}_j\$_2}}$$

Now there are two cases: object c_k is present or is not present in region 2. The first case leads to an infinite computation, as object $\breve{a}_j$ will be situated in region 1 by rule `2r20`, that enforce to applying rule `2r21` or `2r23`. So, object $\$_2$ will be moved in region 1, causing an infinite computation. Consider the second case, i.e., when object c_k is not present in region 2:

$$q_l q_i \check{b}_j \# \boxed{\hat{q}_i\$_1\$_3\boxed{\mathbf{b_j}\mathbf{a_j}\breve{a}_j\$_2}} \Rightarrow^{\texttt{2r2}} q_l q_i \check{b}_j \# \boxed{\hat{q}_i\mathbf{b_j}\mathbf{a_j}\$_1\$_3\boxed{\breve{a}_j\$_2}} \Rightarrow^{\texttt{1r3}}$$

$$q_l q_i \check{\mathbf{b}}_\mathbf{j}\mathbf{b_j}a_j \# \boxed{\hat{q}_i\$_1\$_3\boxed{\breve{a}_j\$_2}} \Rightarrow^{\texttt{1r4}} q_l q_i a_j \# \boxed{\hat{q}_i\check{\mathbf{b}}_\mathbf{j}\mathbf{b_j}\$_1\$_3\boxed{\breve{a}_j\$_2}} \Rightarrow^{\texttt{2r9}}$$

$$q_l q_i a_j \# \boxed{\hat{q}_i\$_1\$_3\boxed{\breve{\mathbf{a}}_\mathbf{j}\check{\mathbf{b}}_\mathbf{j}b_j\$_2}} \Rightarrow^{\texttt{2r22}} q_l q_i a_j \# \boxed{\hat{q}_i\breve{\mathbf{a}}_\mathbf{j}\check{\mathbf{b}}_\mathbf{j}\$_1\$_3\boxed{b_j\$_2}} \Rightarrow^{\texttt{1r9}}$$

$$\mathbf{q_l}\breve{\mathbf{a}}_\mathbf{j} q_i a_j \check{b}_j \# \boxed{\hat{q}_i\$_1\$_3\boxed{b_j\$_2}} \Rightarrow^{\texttt{1r10}} q_i a_j \check{b}_j \# \boxed{q_l\breve{a}_j\hat{q}_i\$_1\$_3\boxed{b_j\$_2}}$$

In this case, q_i is replaced by q_l. Notice that symbols a_j, b_j, $\check{b}_j$, $\hat{q}_i$ have returned to their original positions.

3. END

$$\begin{aligned}
R_{1,f} = {} & \{\texttt{1f1} : (q_f t_3, out),\ \texttt{1f2} : (\$_1 t_3, out),\ \texttt{1f3} : (\$_3 t_3, out),\\
& \quad \texttt{1f4} : (\check{b} t_3, out),\ \texttt{1f5} : (t_3, in)\}\\
& \cup \{\texttt{1f6} : (t_1 t_2, out),\ \ \texttt{1f7} : (t_2 t_4, in),\ \ \texttt{1f8} : (t_4 t_5, out),\ \ \texttt{1f9} : (t_5 t_6, in),\\
& \quad \texttt{1f10} : (t_6 t_7, out),\ \texttt{1f11} : (t_7 t_8, in),\ \texttt{1f12} : (t_8 t_9, out),\ \ \texttt{1f13} : (t_9 F, in)\}.\\
R_{2,f} = {} & \{\texttt{2f1} : (q_f t_1, in)\ , \texttt{2f2} : (q_f t_3, out),\ \texttt{2f3} : (t_1 t_2, out),\\
& \quad \texttt{2f4} : (F, out),\ \texttt{2f5} : (F I_c, in)\}\\
& \cup \{\texttt{2f6} : (I_c b_j, out),\ \texttt{2f7} : (I_c d_j, out) \mid j \in I'\}\\
& \cup \{\texttt{2f8} : (I_c \$_2, out)\}.
\end{aligned}$$

At first, objects $q_f, \$_1, \$_3, \check{b}$ will be moved to the environment by rules `1f1 - 1f4`, and after that all objects b_j, d_j and $\$_2$ will be moved from region 2 to region 1. Hence, in region 2 now there are only the objects c_1 (representing the result of the computation) and only one additional object I_c. □

5 Conclusions

In this paper we proved that any set of natural numbers containing zero generated by symport/antiport P systems with two membranes and minimal

cooperation is finite (for both symport/antiport P systems and for purely symport P systems), while one additional object in the output membrane allows symport/antiport P systems as well as for purely symport P systems with two membranes and minimal cooperation generate any recursively enumerable sets of natural numbers without zero. Thus we improve the result from [1] for symport/antiport P systems with two membranes and minimal cooperation from three objects down to one object and for purely symport P systems from six objects down to one object. Therefore, these results are optimal.

References

1. A. Alhazov, R. Freund, Yu. Rogozhin: Computational Power of Symport/Antiport: History, Advances, and Open Problems. Membrane Computing, International Workshop, WMC 2005, Vienna, 2005, Revised Selected and Invited Papers (R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa, Eds.), *Lecture Notes in Computer Science 3850* (2006) 1–30.
2. A. Alhazov, R. Freund, Yu. Rogozhin: Some Optimal Results on Communicative P Systems with Minimal Cooperation. *Cellular Computing (Complexity Aspects)*, ESF PESC Exploratory Workshop (M.A. Gutiérrez-Naranjo, Gh. Păun, M.J. Pérez-Jiménez, Eds.), Fénix Editora, Sevilla, (2005) 23–36.
3. A. Alhazov, M. Margenstern, V. Rogozhin, Yu. Rogozhin, S. Verlan: Communicative P Systems with Minimal Cooperation. Membrane Computing, International Workshop, WMC 2004, Milan, 2004, Revised Selected and Invited Papers (G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa, Eds.) *Lecture Notes in Computer Science 3365* (2005) 161–177.
4. A. Alhazov, Yu. Rogozhin: Minimal Cooperation in Symport/Antiport P Systems with One Membrane. *Third Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan, Eds.) RGNC TR *01/2005*, University of Seville, Fénix Editora, Sevilla (2005) 29–34.
5. A.Alhazov, Yu.Rogozhin: Towards a Characterization of P Systems with Minimal Symport/Antiport and Two Membranes. In: Pre-proc. of the 7th Workshop on Membrane Computing, WMC7, 17 – 21 July, 2006, Lorentz Center, Leiden (2006) 102–117.
6. A. Alhazov, Yu. Rogozhin, S. Verlan: Symport/Antiport Tissue P Systems with Minimal Cooperation. *Cellular Computing (Complexity Aspects)*, ESF PESC Exploratory Workshop (M.A. Gutiérrez-Naranjo, Gh. Păun, M.J. Pérez-Jiménez, Eds.), Fénix Editora, Sevilla (2005) 37–52.
7. F. Bernardini, M. Gheorghe: On the Power of Minimal Symport/Antiport. *Workshop on Membrane Computing*, WMC 2003 (A. Alhazov, C. Martín-Vide, Gh. Păun, Eds.), Tarragona, 2003, TR *28/03*, Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona (2003) 72–83.
8. F. Bernardini, A. Păun: Universality of Minimal Symport/Antiport: Five Membranes Suffice. Membrane Computing, International Workshop, WMC 2003, Tarragona, Revised Papers (C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, Eds.), *Lecture Notes in Computer Science 2933* (2004) 43–45.
9. R. Freund, M. Oswald: GP Systems with Forbidding Context. *Fundamenta Informaticae 49*, 1–3 (2002) 81–102.
10. R. Freund, M. Oswald: P Systems with Activated/Prohibited Membrane Channels. Membrane Computing International Workshop, WMC-CdeA 02, Curtea de Argeş, 2002. Revised Papers (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, Eds.), *Lecture Notes in Computer Science 2597* (2003) 261–268.

11. R. Freund, A. Păun: Membrane Systems with Symport/Antiport: Universality Results. Membrane Computing International Workshop, WMC-CdeA 02, Curtea de Argeş, 2002. Revised Papers (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, Eds.), *Lecture Notes in Computer Science 2597* (2003) 270–287.
12. P. Frisco: About P Systems with Symport/Antiport. *Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini, Eds), TR *01/2004*, Research Group on Natural Computing, University of Seville (2004) 224–236.
13. P. Frisco, H.J. Hoogeboom: P Systems with Symport/Antiport Simulating Counter Automata. *Acta Informatica 41*, 2–3 (2004) 145–170.
14. P. Frisco, H.J. Hoogeboom: Simulating Counter Automata by P Systems with Symport/Antiport. Membrane Computing International Workshop, WMC-CdeA 02, Curtea de Argeş, 2002. Revised Papers (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, Eds.), *Lecture Notes in Computer Science 2597* (2003) 288–301.
15. L. Kari, C. Martín-Vide, A. Păun: On the Universality of P Systems with Minimal Symport/Antiport Rules. Aspects of Molecular Computing - Essays dedicated to Tom Head on the occasion of his 70th birthday, *Lecture Notes in Computer Science 2950* (2004) 254–265.
16. M. Margenstern, V. Rogozhin, Yu. Rogozhin, S. Verlan: About P Systems with Minimal Symport/Antiport Rules and Four Membranes. *Fifth Workshop on Membrane Computing (WMC5)*, (G. Mauri, Gh. Păun, C. Zandron, Eds.), Universitá di Milano-Bicocca, Milan (2004) 283–294.
17. C. Martín-Vide, A. Păun, Gh. Păun: On the Power of P Systems with Symport Rules, *Journal of Universal Computer Science 8*, 2 (2002) 317–331.
18. M.L. Minsky: *Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey (1967).
19. A. Păun, Gh. Păun: The Power of Communication: P Systems with Symport/Antiport. *New Generation Computing 20* (2002) 295–305.
20. Gh. Păun: Computing with Membranes. *Journal of Computer and Systems Science 61* (2000) 108–143.
21. Gh. Păun: *Membrane Computing. An Introduction.* Springer-Verlag, Berlin (2002).
22. Gh. Păun: Further Twenty Six Open Problems in Membrane Computing (2005). *Third Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan, Eds.) RGNC TR *01/2005*, University of Seville, Fénix Editora, Sevilla (2005) 249–262.
23. Gh. Păun: 2006 Research Topics in Membrane Computing. *Fourth Brainstorming Week on Membrane Computing*, vol. *1* (M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, Eds.), Fénix Edit., Sevilla (2006), 235–251.
24. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages* (3 volumes). Springer-Verlag, Berlin (1997).
25. Gy. Vaszil: On the Size of P Systems with Minimal Symport/Antiport. *Fifth Workshop on Membrane Computing (WMC5)* (G. Mauri, Gh. Păun, C. Zandron, Eds.), Universitá di Milano-Bicocca, Milan (2004) 422–431.
26. S. Verlan: Optimal Results on Tissue P Systems with Minimal Symport/ Antiport. Presented at *EMCC meeting*, Lorentz Center, Leiden (2004).
27. S. Verlan: Tissue P Systems with Minimal Symport/Antiport. Developments in Language Theory, DLT 2004 (C.S. Calude, E. Calude, M.J. Dinneen, Eds), *Lecture Notes in Computer Science 3340*, Springer-Verlag, Berlin (2004) 418–430.
28. P Systems Webpage, `http://psystems.disco.unimib.it`

Expressing Control Mechanisms of Membranes by Rewriting Strategies*

Oana Andrei[1], Gabriel Ciobanu[2,3], and Dorel Lucanu[2]

[1] INRIA-LORIA, Nancy, France
Oana.Andrei@loria.fr
[2] "A.I.Cuza" University of Iaşi, Faculty of Computer Science
[3] Romanian Academy, Institute of Computer Science, Iaşi
{gabriel, dlucanu}@info.uaic.ro

Abstract. In this paper we present a rewriting semantics of membrane systems based on strategies. We use strategies to describe the control mechanisms in membranes. We provide strategies for maximally parallel rewriting, and for maximally parallel rewriting with priorities between rules. Maximally parallel rewriting with promoters or inhibitors requires an additional encoding of the rules.

1 Introduction

In this paper we work with membrane systems (called also P systems) defined in [9]. A membrane consists of a multiset w of objects, a set R of evolution rules, and a control mechanism C describing the way in which the rules are used to modify the multiset w in an evolution step. A very simple way to specify such a membrane is:

```
membrane M
  contents w
  evolution rules R
  control C
end
```

An evolution step of a membrane M modifies its contents w using the evolution rules according to the control mechanism C. We have various control mechanisms in membrane systems inspired by some biological entities. Here we consider the control mechanisms given by maximally parallel rewriting, maximally parallel rewriting with priorities, maximally parallel rewriting with promoters and/or inhibitors. Maximally parallel rewriting means that as many as possible evolution rules are applied in parallel. A (strong) priority relation among rules means that in each region we have a partial order relation on the set of rules, and a rule can be chosen (to process a multiset of objects) only if no rule of a higher priority is applicable in the same region. Promoters and inhibitors formalize the reaction enhancing and reaction prohibiting roles of various substances (molecules)

* This work has been supported by the research grant CEEX 47/2005, Romania.

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 154–169, 2006.

present in cells. In membrane systems, promoters and inhibitors are represented as multisets of objects associated with certain sets of rules. A rule from such a set can be used only if all the promoting objects are present, and none of the inhibiting objects is present in its membrane.

A rewriting system (Σ, A, R) consists of a signature Σ, a set of axioms A, and a set of rewriting rules R. The rewriting process defined by (Σ, A, R) is given by rewriting relation defined by R over Σ-terms modulo the axioms A [8]. A strategy for (Σ, A, R) controls the rewriting process in the sense that only some of the rewriting paths are allowed. Several strategy languages are proposed in the literature [10,7,5].

A membrane M defines a rewriting system (Σ, A, R), where Σ includes the concatenation operator, A includes axioms for associativity, commutativity, and identity of the concatenation, and R includes the evolution rules. In this paper we answer the question whether the control mechanisms in membranes can be described using strategies. We use the strategy language [11] because it includes a complete set of strategy operators and it is independent of the rewriting engine. We also use the Maude implementation presented in [7] to exhibit that the strategies we define for membrane systems behave as expected.

1.1 Examples of Membranes with Priorities and Promoters

We present some examples of membranes implementing arithmetical operations defined over the numbers of objects. More details about other arithmetical operations on numbers represented by using unary and binary compact encodings are described in [4]. Here we emphasize the use of priorities and promoters as control mechanisms in membrane computing, presenting membranes with priorities and promoters for multiplication.

Figure 1 presents a membrane M_1 with priorities for multiplication of n (objects a) by m (objects b), the result being the number of objects d in membrane 0.

```
membrane M1
  contents a^n b^m u
  evolution rules
    ℓ1 : bv →  dev
    ℓ2 : av →  u
    ℓ3 : eu →  dbu
    ℓ4 : au →  v
  control
    pri(ℓ1 > ℓ2, ℓ3 > ℓ4)
end
```

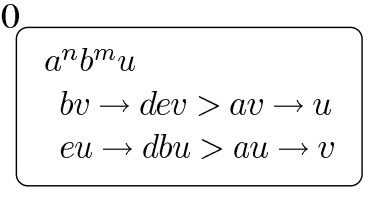


Fig. 1. Multiplication with priorities

In this membrane we use the priority relation between rules; for instance $bv \to dev$ has a higher priority than $av \to u$, meaning the second rule is applied only when the first one cannot be applied anymore. Initially only the rule $au \to v$ can be applied, generating an object v which activates m times the rule $bv \to dev$

(applied in parallel). Then $av \to u$ consumes an a, and transform v into u. Now $eu \to dbu$ is applied m times, followed by another change of u into v by consuming an a (this is done by the rule $au \to v$). The procedure is repeated until no object a is present within the membrane. Note that each time when one object a is consumed, then m objects d are generated. This control mechanism is denoted by $\texttt{pri}(\ell_1 > \ell_2, \ell_3 > \ell_4)$.

Figure 2 presents a membrane M_2 with promoters for multiplication of n (objects a) by m (objects b), the result being the number of objects d in membrane 0. The object a is a promoter in the rule $b \to bd|_a$, i.e., this rule can only be applied in the presence of object a. The available m objects b are used in order to apply m times the rule $b \to bd|_a$ in parallel; based on the availability of a objects, rule $au \to u$ is applied in the same time and consumes an a. The procedure is repeated until no object a is present within the membrane. Note that each time when one object a is consumed, then m objects d are generated. This control mechanism is denoted by $\texttt{prom}(\ell_1, \ell_2)$.

```
membrane M2
  contents a^n b^m u
  evolution rules
    ℓ1 : b →  bd|a
    ℓ2 : au →  u
  control
    prom(ℓ1, ℓ2)
end
```

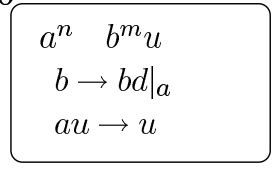


Fig. 2. Multiplication with promoters

2 Strategies in Rewriting Systems

In general terms, a strategy is setting the objective(s) of a computation. In term rewriting systems, a strategy is an expression s involving rewriting rules and strategy operators. The objectives of s are *strategic transitions* $t \xrightarrow{s} t'$, where t and t' are terms. The tactics indicate how we are supposed to reach the objectives of a strategy. A tactic of a strategic transition $t \xrightarrow{s} t'$ is a rewriting sequence $t = t_0 \to \cdots \to t_n = t'$ denoted shortly by $t \rightsquigarrow t'$ which applies the rewriting rules according to a strategy s. We write $t \rightsquigarrow_n t'$ when we want to specify the length of the rewriting sequence.

A rewriting strategy controls the rewriting process in the sense that only some of the rewriting paths are allowed, namely those given by tactics. For instance, computation of the arithmetical expressions involving associative and commutative $+$ and $*$, without parenthesis, could follow several pathways.

$$2 + 3 * 5 \to 2 + 15 \to 17$$
$$2 + 3 * 5 = 2 + 5 * 3 \to 2 + 15 \to 17$$
$$2 + 3 * 5 \to 5 * 5 \to 25$$
$$2 + 3 * 5 = 2 + 5 * 3 = 5 + 2 * 3 \to 5 + 6 \to 11$$

Only some of them are correct according to the usual arithmetical rules (for the above example, these are the pathways leading to 17). We aim to define a strategy `eval` such that $2 + 3 * 5 \xrightarrow{\texttt{eval}} 17$. In general, a rewriting strategy language consists of expressions s constructed with rewriting rules (viewed as basic strategies) and strategy operators such that $expr \xrightarrow{s} expr'$. Here we use a strategy language inspired by [11]. In this language, `eval` can be expressed as `repeat(multiply` $+$ `add)`. This means that we apply multiplication repeatedly until it is not possible anymore, followed by addition. In the sequel we provide the definition of the strategy language and illustrate it by examples.

Basic strategies. Each evolution rule $\ell : u \to v$ defines a strategy operator with the operational semantics given by the strategic transition $w \xrightarrow{\ell} w'$, where $w = u$, $w' = v$, and these equalities are modulo associativity, commutativity, and identity. The only rewriting tactic corresponding to such a strategic transition is $w \rightsquigarrow_1 w'$, i.e., the rewriting of length one defined by the rule. The uniqueness of the tactic is given by the fact that the evolution rules have not variables.

Identity. We consider a strategy operator `id` with the operational semantics given by $w \xrightarrow{\texttt{id}} w$, where w is a multiset. The only rewriting tactic corresponding to such a strategic transition is $w \rightsquigarrow_0 w$, i.e., the rewriting of length zero.

Congruence. Each operator name (term constructor) defines a strategy operator whose parameter-strategies are applied to its arguments. Since in membrane systems we have only one operator, namely the associative and commutative concatenation, we use a specific strategy operator `mset` with a variable number of parameter-strategies. The operational semantics of the operator `mset` is

$$\frac{w_1 \xrightarrow{s_1} w'_1 \quad \dots \quad w_n \xrightarrow{s_n} w'_n}{w_1 \cdots w_n \xrightarrow{\texttt{mset}(s_1,\dots,s_n)} w'_1 \cdots w'_n}$$

Remark 1. The concatenation operator `__` has variable arity due to its associativity and identity laws. For instance, $aabbb$ can be written as `__`(aa, bbb), `__`(a, a, b, b, b), `__`(a, ab, bb), and so on. The notation `__`[`assoc comm id`: ε] intends to capture this feature together with commutativity of `__`. In order to avoid any confusion, we prefer to denote this variadic operator by `mset`. Therefore we use `mset`$(s_1, \dots, s_n)$ instead of `__`[`assoc comm id`: ε]$(s_1, \dots, s_n)$. The general case of congruence provided by operators with attributes is discussed in [3].

If $w_i \rightsquigarrow w'_i$ is a rewriting tactic of $w_i \xrightarrow{s_i} w'_i$ for $i = 1, \dots, n$, then $w_1 \cdots w_n \rightsquigarrow w'_1 \cdots w_n \rightsquigarrow \cdots \rightsquigarrow w'_1 \cdots w'_n$ is a tactic of $w_1 \cdots w_n \xrightarrow{\texttt{mset}(s_1,\dots,s_n)} w'_1 \cdots w'_n$. Since the concatenation is associative and commutative, the rewriting tactics can be combined in an arbitrary order.

Example 1. Considering the rules in Figure 1, we have $abu \xrightarrow{\texttt{mset}(\ell_4,\mathrm{id})} bv$ because $au \xrightarrow{\ell_4} v$, $b \xrightarrow{\mathrm{id}} b$, $abu = aub$, and $vb = bv$.

Sequential composition. $s_1; s_2$ applies s_1 and, if it succeeds, then it applies s_2. The operational semantics of $s_1; s_2$ is given by

$$\frac{w \xrightarrow{s_1} w' \quad w' \xrightarrow{s_2} w''}{w \xrightarrow{s_1;s_2} w''}$$

If $w \rightsquigarrow w'$ is a rewriting tactic of $w \xrightarrow{s_1} w'$ and $w' \rightsquigarrow w''$ is a rewriting tactic of $w' \xrightarrow{s_2} w''$, then $w \rightsquigarrow w' \rightsquigarrow w''$ is a rewriting tactic of $w \xrightarrow{s_1;s_2} w''$.

Example 2. Since $abu \xrightarrow{\texttt{mset}(\ell_4,\mathrm{id})} bv$ and $bv \xrightarrow{\ell_1} dev$, we have

$$abu \xrightarrow{\texttt{mset}(\ell_4,\mathrm{id});\ell_1} dev.$$

Non-deterministic choice. $s_1 + s_2$ chooses between the strategies s_1 and s_2 such that the chosen strategy succeeds. The operational semantics of $s_1 + s_2$ is given by

$$\frac{w \xrightarrow{s_1} w'}{w \xrightarrow{s_1+s_2} w'} \qquad \frac{w \xrightarrow{s_2} w'}{w \xrightarrow{s_1+s_2} w'}$$

If $w \rightsquigarrow w'$ is a rewriting tactic of $w \xrightarrow{s_1} w'$ or $w \xrightarrow{s_2} w'$, then $w \rightsquigarrow w'$ is a rewriting tactic of $w \xrightarrow{s_1+s_2} w'$.

Example 3. Considering the evolution rules in Figure 2, we have $abu \xrightarrow{\ell_1+\ell_2} bu$ or $abu \xrightarrow{\ell_1+\ell_2} abdu$.

Definition 1. *We say that a strategy* s fails *on* w *iff there is no* w' *such that* $w \xrightarrow{s} w'$. *We write* $w \xrightarrow{s} \uparrow$.

In other words, $w \xrightarrow{s} \uparrow$ means that for any w', $w \xrightarrow{s} w'$ has no rewriting tactic. The above definition applies to all the strategies of the language.

Deterministic choice. $s_1 \leftarrow\!\!+ s_2$ chooses the left argument first; the second strategy is considered if the first strategy fails. The operational semantics of $s_1 \leftarrow\!\!+ s_2$ is given by

$$\frac{w \xrightarrow{s_1} w'}{w \xrightarrow{s_1 \leftarrow\!\!+ s_2} w'} \qquad \frac{w \xrightarrow{s_1} \uparrow \quad w \xrightarrow{s_2} w''}{w \xrightarrow{s_1 \leftarrow\!\!+ s_2} w''}$$

If $w \rightsquigarrow w'$ is a rewriting tactic of $w \xrightarrow{s_1} w'$, then $w \rightsquigarrow w'$ is a rewriting tactic of $w \xrightarrow{s_1 \leftarrow\!\!+ s_2} w'$. If $w \xrightarrow{s_1} w'$ has no rewriting tactic for any w', then any rewriting tactic of $w \xrightarrow{s_2} w''$ is a rewriting tactic of $w \xrightarrow{s_1 \leftarrow\!\!+ s_2} w''$.

Example 4. Using again the evolution rules in Figure 1, we have $abv \xrightarrow{\ell_1 \leftarrow\!\!+ \ell_2} adev$. We cannot deduce $abv \xrightarrow{\ell_1 \leftarrow\!\!+ \ell_2} bu$ because ℓ_1 succeeds on abv. On the other hand, ℓ_1 fails on $adev$ and so we have $adev \xrightarrow{\ell_1 \leftarrow\!\!+ \ell_2} deu$.

Strategy definition. A *strategy definition* is an expression $\varphi(z_1, \ldots, z_n) \stackrel{\text{def}}{=} s$, where any free variable in s belongs to $\{z_1, \ldots, z_n\}$, and φ is a strategy identifier. The operational semantics is given by

$$\frac{w \xrightarrow{s[z_1:=s_1,\ldots,z_n:=s_n]} w'}{w \xrightarrow{\varphi(s_1,\ldots,s_n)} w'} \text{ if } \varphi(z_1, \ldots, z_n) \stackrel{\text{def}}{=} s$$

where $s[z_1 := s_1, \ldots, z_n := s_n]$ is the strategy expression obtained from s by replacing the free occurrences of the variables z_i with s_i. Each rewriting tactic of $w \xrightarrow{s[z_1:=s_1,\ldots,z_n:=s_n]} w'$ is a rewriting tactic of $w \xrightarrow{\varphi(s_1,\ldots,s_n)} w'$.

Fixpoint operator. The fixpoint operator $\mu z(s)$ allows to define strategies that repeatedly apply a certain strategy s. For instance, the strategy `repeat`, which applies s as many times as possible, is defined as

$$\texttt{repeat}(s) \stackrel{\text{def}}{=} \mu z((s; z) \leftarrow \mathsf{id})$$

The operational semantics of $\mu z(s)$ is given by

$$\frac{w \xrightarrow{s[z:=\mu z(s)]} w'}{w \xrightarrow{\mu z(s)} w'}$$

The fixpoint operator μ binds any occurrence of variable z in strategy s.
Each rewriting tactic of $w \xrightarrow{s[z:=\mu z(s)]} w'$ is a rewriting tactic of $w \xrightarrow{\mu z(s)} w'$.

3 Strategy Semantics of Control Mechanisms

In membrane systems, a computation step $w \Rrightarrow w'$ can be presented as a transition from a contents to another contents according to a control mechanism involving priorities and/or promoters and/or inhibitors. The computing engine is represented by the maximal parallel rewriting. When we refer to a sequential implementation for membrane computing, such a computation is translated in sequential rewritings. Such a sequential implementation based on rewritings is presented in [1]. In this paper, we consider the membrane computation steps as objectives provided by strategies, and their sequential implementations as tactics of these strategies.

We investigate whether we can find a strategy s such that $w \Rrightarrow w'$ iff $w \xrightarrow{s} w'$. We give a strategic semantics for maximal parallel rewriting, as well as for maximal parallel rewriting with priorities between rules. However we find that a more powerful mechanism than strategies is needed to provide semantics for maximal rewriting with promoters or inhibitors. A useful encoding of the rules can solve this problem, and finally we can provide the semantics for maximal rewriting of membrane systems involving promoters/inhibitors. The encoding can be used in a uniform way to provide the strategy semantics for simple maximal rewriting, maximal rewriting of membrane systems with priorities, and maximal rewriting of systems with promoters/inhibitors.

3.1 Strategic Semantics of Maximal Parallel Rewriting

Let R be a set of evolution rules, and w a multiset of objects. If $\ell : u \to v$ is an evolution rule in R, then w is *ℓ-irreducible* if $w \xrightarrow{\texttt{mset}(\ell,\mathrm{id})}\uparrow$. In other words, w is ℓ-irreducible iff there is no w' such that $w \to w'$ applying the rule labelled by ℓ. Moreover, w is *R-irreducible* if w is ℓ-irreducible for all $\ell : u \to v \in R$. We say that w is *maximally parallel rewritten* in w' iff $w = u_1 \cdots u_n z$, $w' = v_1 \cdots v_n z$, $\ell_i : u_i \to v_i$ is a rule in R for $i = 1, \ldots, n$, $n > 0$, and z is R-irreducible. We write $w \Rrightarrow_R w'$.

Given a set $R = \{\ell_i : u_i \to v_i \mid 1 \leq i \leq n\}$ of evolution rules, we define a strategy

$$\begin{aligned} &mpr \stackrel{\text{def}}{=} \mu x(s_1 + \cdots + s_n) \\ &\text{where } s_i = \texttt{mset}(\ell_i, x \leftarrow \mathrm{id}), \text{ for } i = 1, \ldots, n. \end{aligned}$$

Since the definition of this strategy mpr depends on R, we prefer to write it in an equivalent form

$$mpr(R) \stackrel{\text{def}}{=} \texttt{mset}(\ell_1, mpr(R) \leftarrow \mathrm{id}) + \cdots + \texttt{mset}(\ell_n, mpr(R) \leftarrow \mathrm{id}).$$

If $R = \emptyset$, then $w \xrightarrow{mpr(\emptyset)}\uparrow$ for any w.

Theorem 1. *Given a set R of evolution rules, then*

$$w \Rrightarrow_R w' \text{ iff } w \xrightarrow{mpr(R)} w'.$$

Proof. We first assume that $w \Rrightarrow_R w'$. We have $w = u_{i_1} \cdots u_{i_k} z$, $w' = v_{i_1} \cdots v_{i_k} z$, rules $\ell_{i_j} : u_{i_j} \to v_{i_j}$ from $R = \{\ell_i : u_i \to v_i \mid i = 1, \ldots, n\}$ for $j = 1, \ldots, k$, $k > 0$, and z is R-irreducible. We prove $w \xrightarrow{mpr(R)} w'$ by induction on k.

If $k = 1$, then the proof is:

$$\dfrac{\dfrac{\dfrac{\dfrac{z \xrightarrow{mpr(R)}\uparrow \qquad z \xrightarrow{\mathrm{id}} z}{z \xrightarrow{mpr(R)+\mathrm{id}} z} \qquad u_i \xrightarrow{\ell_i} v_i}{u_i z \xrightarrow{s_i} v_i z}}{u_i z \xrightarrow{s_1+\cdots+s_n} v_i z}}{u_i z \xrightarrow{mpr(R)} v_i z}$$

where $i = i_1$. If $k > 1$, then $u_{i_k} z \xrightarrow{mpr(R)} v_{i_k} z$ as above, and $u_1 \ldots u_{i_{k-1}} \xrightarrow{mpr} v_1 \ldots v_{i_{k-1}}$ by the inductive hypothesis. We get $w \xrightarrow{mpr(R)} w'$ by the definition of the fixpoint operator, and by the fact that the concatenation in the left hand side does not produce new reducible multisets.

Conversely, if $w \xrightarrow{mpr(R)} w'$ then we prove $w \Rrightarrow_R w'$ by induction on the depth of the proof tree. By the definition of $mpr(R)$, we have rules $\ell_i : u_i \to v_i$ in R, w_i and w_i' such that $w = u_i w_i$, $w' = v_i w_i'$, and $w_i \xrightarrow{mpr(R)} w_i'$. By the inductive hypothesis we have $w_i \Rrightarrow_R w_i'$, and we get $w = u_i w_i \Rrightarrow_R v_i w_i' = w'$ by the definition of $\Rrightarrow$. □

Example 5. If R consists of rules $\ell_1 : ab \to c$ and $\ell_2 : bb \to d$, then the inference tree for $aabbb \xrightarrow{mpr} cda$ is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{a \xrightarrow{mpr(R)} \uparrow \qquad a \xrightarrow{\mathrm{id}} a}{a \xrightarrow{mpr(R)+\mathrm{id}} a} \qquad bb \xrightarrow{\ell_2} d}{abb \xrightarrow{s_2} da}}{abb \xrightarrow{s_1+s_2} da}}{abb \xrightarrow{mpr(R)} da}}{abb \xrightarrow{mpr(R)+\mathrm{id}} da} \qquad ab \xrightarrow{\ell_1} c}{aabbb \xrightarrow{s_1} cda}}{aabbb \xrightarrow{s_1+s_2} cda}}{aabbb \xrightarrow{mpr(R)} cda}$$

```
membrane M
  contents a a b b b
  evolution rules
    ℓ1 : a b -> c
    ℓ2 : b b -> d
  control
    mpr(ℓ1, ℓ2)
end
```

is represented in Maude strategy language [7] as follows:

```
( mod MM is
    including PSCONFIGURATION .
    op M : -> Label .
    ops a b c d : -> Obj .
    vars W W' W'' : Soup .
    var L : Label .
    op contents : -> Membrane .
    eq contents = < M | a a b b b > .

    rl [l1] : a b => c .
    rl [l2] : b b => d .
  endm )

( stratdef MM-STRAT is
    strat mpr = ( matchrew W' W'' by W' using top(l2) ,
                                     W'' using (mpr orelse idle) ) |
          (       matchrew W' W'' by W' using top(l1) ,
                                     W'' using (mpr orelse idle) ) .
  endsd )
```

The sorts `Obj, Soup, Label, Membrane`, and the concatenation operator are defined in the module `PSCONFIGURATION`. Concatenation is denoted by `__`, and it is declared with the attributes associativity, commutativity, and identity [6].

The strategy $s_i = \texttt{mset}(\ell_i, mpr(R) \Lsh \mathrm{id})$ is represented by the expression `matchrew W' W'' by W' using top(`ℓ_i`) , W'' using (mpr orelse idle)`, and $s_1 + s_2$ is represented by $s_1 \mid s_2$. We use `srewall` command in order to see how the contents is modified using `mpr` strategy. This command performs the all rewritings supplied by a given strategy.

```
Maude> (srewall contents using matchrew  < L | W > by W using mpr .)
rewrites: 4079 in 27ms cpu (27ms real) (145699 rewrites/second)
rewrite with strategy :
Solution 1 :    < M | d a c >
Solution 2 :    < M | b c c >
Maude>
```

3.2 Strategic Semantics of Maximal Parallel Rewriting with Priorities

Let R be a set of evolution rules together with a partial order $\succ$. If $\ell \succ \ell'$, then we say that ℓ has a greater priority than ℓ'. An evolution rule is applied in an evolution step only if no rule of a higher priority can be applied.

Definition 2. *Let R be a set of evolution rules together with a priority relation $\succ$. We say that w is* maximally parallel rewritten *in w' w.r.t. R, and write $w \Rrightarrow_R w'$, iff $w \Rrightarrow_{max(R,w)} w'$, where $max(R, w)$ represents the highest priority evolution rules of R which are applicable to w.*

Note that $max(R, w)$ is a discrete partial ordered set, and $w \Rrightarrow_{max(R,w)} w'$ is defined as in 3.1. However, we cannot apply the strategy $mpr(max(R, w))$ because it depends on multiset w (usually a strategy is independent of the multiset).

Definition 3. *Let R be a set of evolution rules together with a priority relation $\succ$ such that $\{\ell_i : u_i \to v_i \mid i = 1, \ldots, n\}$ is the subset of the rules with maximal priority. Then the strategy $pri(R)$ is defined as follows:*

$$\begin{aligned} &pri(R) \stackrel{def}{=} s_1 + \cdots + s_n, \\ &s_i = \texttt{mset}(\ell_i, pri(filter(R, \ell_i)) \Lsh id) \Lsh pri(R \setminus \{l_i\}) \\ &\text{for } i = 1, \ldots, n, \end{aligned}$$

where $filter(R, \ell_i)$ is obtained from R by removing the rules having lower priority than ℓ_i. If $R = \emptyset$, then $w \xrightarrow{pri(\emptyset)} \uparrow$.

Example 6. Let us suppose that R consists of $\ell_1 : ab \to c \succ \ell_2 : bb \to d$. We have:

$$\begin{aligned} &pri(R) \stackrel{\text{def}}{=} s_1 \quad (\ell_1 \text{ is the only maximal element in } R) \\ &s_1 = \texttt{mset}(\ell_1, pri(\ell_1) \Lsh \mathrm{id}) \Lsh pri(\ell_2) \\ &pri(\ell_1) \stackrel{\text{def}}{=} \texttt{mset}(\ell_1, pri(\ell_1) \Lsh \mathrm{id}) \\ &pri(\ell_2) \stackrel{\text{def}}{=} \texttt{mset}(\ell_2, pri(\ell_2) \Lsh \mathrm{id}) \end{aligned}$$

The inference tree for $aabbb \Rrightarrow ccb$ is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{b \xrightarrow{pri(\ell_1)} \uparrow \qquad b \xrightarrow{\mathrm{id}} b}{b \xrightarrow{pri(\ell_1)+\mathrm{id}} b} \qquad ab \xrightarrow{\ell_1} c}{abb \xrightarrow{\mathtt{mset}(\ell_1, pri(\ell_1)+\mathrm{id})} cb}}{abb \xrightarrow{pri(\ell_1)} cb}}{abb \xrightarrow{pri(\ell_1)+\mathrm{id}} cb} \qquad ab \xrightarrow{\ell_1} c}{aabbb \xrightarrow{\mathtt{mset}(\ell_1, pri(\ell_1)+\mathrm{id})} ccb}}{aabbb \xrightarrow{\mathtt{mset}(\ell_1, pri(filter(R,\ell_1))+\mathrm{id})} ccb}}{aabbb \xrightarrow{pri(R)} ccb}$$

```
membrane M
  contents a a b b b
  evolution rules
    ℓ1 : a b -> c
    ℓ2 : b b -> d
  control
    pri(ℓ1 ≻ ℓ2)
end
```

is represented in Maude strategy language as follows:

```
( mod MM is
    including PSCONFIGURATION .
    op M : -> Label .
    ops a b c d : -> Obj .
    vars W W' W'' : Soup .
    var L : Label .
    op contents : -> Membrane .
    eq contents = < M | a  a  b  b  b > .

    rl [l1] : a  b => c .
    rl [l2] : b  b => d .
  endm)

( stratdef MM-STRAT is
    strat pri1 = matchrew W' W'' by W' using top(l1) ,
                                    W'' using ( pri1 orelse idle ) .
    strat pri2 = matchrew W' W'' by W' using top(l2) ,
                                    W'' using ( pri2  orelse idle ) .
    strat pri = pri1 orelse pri2 .
  endsd)
```

Even rule `l2` matches the contents, it cannot be used because `l1` has a higher priority:

```
Maude> (srewall contents using xmatchrew < L | W > by W using pri .)
rewrites: 1530 in 10ms cpu (10ms real) (139116 rewrites/second)
rewrite with strategy :
Solution 1 :    < M | b c c >
Maude>
```

Theorem 2. *Given a set R of evolution rules together with a priority relation $\succ$,*
$$w \Rrightarrow_R w' \text{ iff } w \xrightarrow{pri(R)} w'.$$

Proof. (Sketch) The main idea is similar to that in the proof of Theorem 1. The correct handling of the priorities is assured by the following facts:

- only rules with maximal priority are applied,
- once a rule is applied, all the rules having smaller priorities are removed from the current set of rules by using $filter$ operator, and
- if a rule with a maximal priority cannot be applied, then it is removed. □

3.3 Strategic Semantics of Maximal Parallel Rewriting with Promoters

An *evolution rule with promoter* is a rewriting rule of the form $\ell : u \to v|_p$, where the promoter p does not necessarily occur in u. Such a rule can be applied in an evolution step $w \Rrightarrow w'$ only if w contains both u and p. Note that a single occurrence of a promoter can be used by more than one rule even the promoter can be consumed by some other rule. The presence of the promoter makes it possible to use a rule with promoter as many times as possible, without any restriction.

Let R consist of the following rules with promoters: $\ell_1 : aq \to c|_p$ and $\ell_2 : bp \to d|_q$. Consider s a strategy applying the rules R over $abpq$. If s applies first ℓ_1, then the information that promoter p was present in the initial contents is lost, and ℓ_2 cannot be applied anymore. If s applies first ℓ_2, then the information that promoter q was present in the initial contents is lost, and ℓ_1 cannot be applied anymore. Therefore we claim that there is no strategy expressed in terms of rules R and the existing strategy operators which can implement the maximal parallel rewriting with promoters. A more powerful mechanism is needed. We show that an encoding of the membrane specification together with the strategies defined over the corresponding encoded rules are enough for implementing the maximal rewriting with promoters.

Given a set R of evolution rules with or without promoters, we construct a set $\widehat{R}$ of rewrite rules and a strategy $prom(\widehat{R})$ such that $w \Rrightarrow_R w'$ iff $w \xrightarrow{prom(\widehat{R})} w'$. Let $pset(R, w)$ denote the set of promoters occurring in w w.r.t. the set of rules R. The set $\widehat{R}$ consists of the following rules:

- $\texttt{compute} : w \to \varepsilon(w, pset(R, w))$,
- $\texttt{forget} : w'(w, s) \to w'w$, together with
- a rule $\hat{\ell} : w'(wu, s) \to w'v(w, s)$ for each rule $\ell : u \to v$ without promoter, and
- a rule $\hat{\ell} : w'(wu, ps) \to w'v(w, ps)$ for each rule $\ell : u \to v|_p$ with promoter,

where w', w range over multisets, and s ranges over the sets of promoters.

The rule `compute` stores the set of promoters occurring in w as the second component of the pair, and this component remains unchanged during the application of the evolution rules. This information is used by the rules with promoters: such a rule is applied only if its promoter is present in the second component. Note that the processed part w' lies in the front of the pair (w, s), and it is not affected by the next evolution rules applied in the current step; the evolution rules consumes only objects from the first component w of the pair.

Definition 4. *We suppose that* $R = \{\ell_i : u_i \to v_i \mid i = 1, \ldots, n\}$*, and let* $\widehat{R}$ *be computed as above. Then the strategy* $prom(\widehat{R})$ *is*

$$prom(\widehat{R}) \stackrel{def}{=} \texttt{compute}; \texttt{repeat}(\hat{\ell}_1 + \cdots + \hat{\ell}_n); \texttt{forget}$$

Example 7. For R given above, $\widehat{R}$ consists of `compute`, `forget`, together with
$\hat{\ell}_1 : w'(aqw, ps) \to w'c(w, ps)$ and
$\hat{\ell}_2 : w'(bpw, qs) \to w'd(w, ps)$.

The inference tree of $abpq \xrightarrow{prom(\widehat{R})} cd$ is obtained as follows:

T_1:

$$\frac{(abpq, pq) \xrightarrow{\hat{\ell}_1} c(bp, pq)}{(abpq, pq) \xrightarrow{\hat{\ell}_1 + \hat{\ell}_2} c(bp, pq)}$$

T_2:

$$\frac{\dfrac{c(bp, pq) \xrightarrow{\hat{\ell}_2} cd(\varepsilon, pq)}{c(bp, pq) \xrightarrow{\hat{\ell}_1 + \hat{\ell}_2} cd(\varepsilon, pq)} \qquad cd(\varepsilon, pq) \xrightarrow{\hat{\ell}_1 + \hat{\ell}_2} \uparrow}{c(bp, pq) \xrightarrow{\texttt{repeat}(\hat{\ell}_1 + \hat{\ell}_2)} cd(\varepsilon, pq)}$$

T_3:

$$\frac{abpq \xrightarrow{\texttt{compute}} (abpq, pq) \qquad \dfrac{\dfrac{T_1 \qquad T_2}{(abpq, pq) \xrightarrow{(\hat{\ell}_1 + \hat{\ell}_2); \texttt{repeat}(\hat{\ell}_1 + \hat{\ell}_2)} cd(\varepsilon, pq)}}{(abpq, pq) \xrightarrow{\texttt{repeat}(\hat{\ell}_1 + \hat{\ell}_2)} cd(\varepsilon, pq)}}{abpq \xrightarrow{\texttt{compute}; \texttt{repeat}(\hat{\ell}_1 + \hat{\ell}_2)} cd(\varepsilon, pq)}$$

Finally,

$$\frac{T_3 \qquad cd(\varepsilon, pq) \xrightarrow{\texttt{forget}} cd}{abpq \xrightarrow{prom(\widehat{R})} cd}$$

We have the following encoding:

```
membrane M
  contents a a b
  evolution rules
    ℓ1 : a q -> c |p
    ℓ2 : b p -> d |q
  control
    prom(ℓ1, ℓ2)
end
```

⇒

```
rewsystem M̂
  rewriting rules
   compute : W → ε (W, p q)
   forget : W'(W, p q) → W' W
   ℓ̂1 : W'(W a q, p W'') → W' c(W, p W'')
   ℓ̂2 : W'(W b p, p W'') → W' d(W, p W'')
  strategy
    prom(ℓ̂1, ℓ̂2)
end
```

The encoded membrane $\widehat{M}$ is represented in Maude strategy language as follows:

```
( mod MM is
    including PSCONFIGURATION .
    op M : -> Label .
    ops a b c d p q : -> Obj .
    vars S S' W W' X Y : Soup .
    var L : Label .
    var ES : EncodedSoup .
    op contents : -> Membrane .
    eq contents = < M | a b p q > .

    rl [compute] : W => empty ( W , p q ) .
    rl [forget] : W' ( W , p q ) => W' W .

    rl [l1] : W' ( W a q , p W" ) => (W' c) ( W , p W" ) .
    rl [l2] : W' ( W b p , q W" ) => (W' d) ( W , q W" ) .
  endm
)

( stratdef MM-STRAT is
    strat prom = top(compute) ; ( l1 | l2 ) ! ; top(forget) .
  endsd)
```

The module `PSCONFIGURATION` additionally includes a sort `EncodedSoup` and an operator:

```
  op _`(_`,_`) : Soup Soup Soup -> EncodedSoup .
```

`repeat(_)` is represented in Maude strategy language by `_!`.
The strategy `prom` works properly:

```
Maude> (srewall contents using xmatchrew < L | W > by W using prom .)
rewrites: 766 in 8ms cpu (9ms real) (85120 rewrites/second)
rewrite with strategy :
Solution 1 :    < M | d c >
Maude>
```

Theorem 3. *Given a set R of evolution rules with promoters, then*

$$w \Rrightarrow_R w' \text{ iff } w \xrightarrow{prom(\widehat{R})} w'.$$

Proof. (Sketch) $w \Rrightarrow_R w'$ iff $(w, pset(R,w)) \xrightarrow{\texttt{repeat}(\hat{\ell}_1+\cdots+\hat{\ell}_n)} (w', pset(R,w))$. If $w \Rrightarrow_R w'$, then it follows that $w = u_{i_1}\cdots u_{i_k}z$, $w' = v_{i_1}\cdots v_{i_k}z$, either $\ell_{i_j} : u_{i_j} \to v_{i_j}$ or $\ell_{i_j} : u_{i_j} \to v_{i_j}|_p$ is a rule in $R = \{\ell_i : u_i \to v_i \mid i = 1,\ldots,n\}$ for $j = 1,\ldots,k$, $k > 0$, and z is R-irreducible. If $\ell_{i_j} : u_{i_j} \to v_{i_j}|_p$ is a rule involving a promoter p, then p occurs in w and therefore it belongs to $pset(R,w)$. We prove that $w \xrightarrow{prom(\widehat{R})} w'$ by induction on k.

Conversely, if $(w, pset(w)) \xrightarrow{\texttt{repeat}(\hat{\ell}_1+\cdots+\hat{\ell}_n)} (w', pset(R,w))$, then we prove $w \Rrightarrow_R w'$ by induction on the depth of the inference tree. □

3.4 Strategic Semantics of Maximal Parallel Rewriting with Inhibitors

An *evolution rule with inhibitor* is a rewriting rule of the form $\ell : u \to v|_{\neg p}$. Such a rule can be applied in an evolution step $w \Rrightarrow w'$ only if the inhibitor p is not present in w.

We proceed in a similar way as for promoters. Let A be the set of all the objects and $iset(A, w)$ be the complement w.r.t. A of the set of inhibitors occurring in w. The encoding uses $iset(A, w)$ instead of $pset(R, w)$, and rules with inhibitors instead of rules with promoters. The set $\widehat{R}$ consists of the following rules:

- $\texttt{compute} : w \to (w, iset(A, w))$,
- $\texttt{forget} : w'(w, s) \to w'w$, together with
- a rule $\hat{\ell} : w'(wu, s) \to w'v(w, s)$ for each rule $\ell : u \to v$ without inhibitor, and
- a rule $\hat{\ell} : w'(wu, ps) \to w'v(w, ps)$ for each rule $\ell : u \to v|_{\neg p}$ with inhibitor,

where w', w range over multisets, and s ranges over the sets of inhibitors.

Definition 5. *We suppose that* $R = \{\ell_i : u_i \to v_i \mid i = 1, \ldots, n\}$*, and let* $\widehat{R}$ *be computed as above. Then the strategy* $inhib(\widehat{R})$ *is*

$$inhib(\widehat{R}) \stackrel{def}{=} \texttt{compute}; \texttt{repeat}(\hat{\ell}_1 + \cdots + \hat{\ell}_n); \texttt{forget}$$

Theorem 4. *Given a set* R *of evolution rules with inhibitors, then*

$$w \Rrightarrow_R w' \text{ iff } w \xrightarrow{inhib(\widehat{R})} w'.$$

3.5 Strategic Semantics of Maximal Parallel Rewriting with Promoters and Inhibitors

When we have rules involving both promoters and inhibitors, we encode a multiset by a triple $(w, pset(R, w), iset(A, w))$ in order to have information about both promoters and inhibitors. The set $\widehat{R}$ consists of the following rules:

- $\texttt{compute} : w \to (w, pset(R, w), iset(A, w))$,
- $\texttt{forget} : w'(w, s, s') \to w'w$, together with
- a rule $\hat{\ell} : w'(wu, s, s') \to w'v(w, s, s')$ for each rule $\ell : u \to v$ in R without promoter or inhibitor,
- a rule $\hat{\ell} : w'(wu, ps, s') \to w'v(w, ps, s')$ for each rule $\ell : u \to v|_p$ in R with promoter, and
- a rule $\hat{\ell} : w'(wu, s, ps') \to w'v(w, s, ps')$ for each rule $\ell : u \to v|_{\neg p}$ in R with inhibitor.

This encoding is general, and it can be used even one or both sets of promoters and inhibitors are empty.

Definition 6. *We suppose that* $R = \{\ell_i : u_i \to v_i \mid i = 1, \ldots, n\}$*, and let* $\widehat{R}$ *be computed as above. Then the strategy* $prominhib(\widehat{R})$ *is*

$$prominhib(\widehat{R}) \stackrel{def}{=} \texttt{compute}; \texttt{repeat}(\hat{\ell}_1 + \cdots + \hat{\ell}_n); \texttt{forget}$$

Theorem 5. *Given a set* R *of evolution rules involving eventually promoters and inhibitors, then*

$$w \Rrightarrow_R w' \text{ iff } w \xrightarrow{prominhib(\widehat{R})} w'.$$

4 Conclusion

The main contribution of the paper is given by the use of strategies in defining a term rewriting semantics of the membrane systems. Comparing with the operational semantics presented in [2], here we use strategies to describe the control mechanisms in membranes. We give a strategic semantics for maximal parallel rewriting, as well as for maximal parallel rewriting with priorities between rules. However we find that a more powerful mechanism than strategies is needed to provide semantics for maximal rewriting with promoters or inhibitors. A useful encoding of the rules can solve this problem, and finally we can provide the semantics for maximal rewriting of membrane systems involving promoters and inhibitors. In this way, term rewriting semantics distinguishes between priorities and promoters/inhibitors, namely it is easier to describe priorities than promoters/inhibitors.

We adapt the strategy language presented in [10], taking into consideration that:

- only a part of the strategy operators defined in [10] is used for describing the control mechanisms in membranes;
- since the membranes work over multisets of objects, the strategy congruence operators are extended to handle the associativity and commutativity.

In the paper we consider elementary membranes. The approach can be extended to more complex membrane systems. Such a membrane system is a hierarchical structure of membranes which could be specified as follows:

```
system Π
  objects A
  membrane M1
    contents w1
    evolution rules R1
    control C1
  end
  membrane M2
    contents w2
    evolution rules R2
    control C2
  end
```

```
  ...
  structure [M1[M2...]
end
```

Usually a global clock is assumed [9]. At each tick of this clock, the current configuration of the system is transformed into another one in three phases: evolution - where all the membranes evolve according to their rules and control mechanisms, communication - where objects are exchanged between adjacent membranes, and membrane dissolving. The configurations of a membrane system Π can be described as terms in an appropriate algebraic specification [2]. Encoding all the membranes as rewriting systems with strategies, we can use traversal operators like `bottomup` or `topdown` [10,7] over configurations in order to extend the evolution of each membrane to the global evolution phase of a transition step. We may proceed in the same way for the other two phases. Consequently, a transition step is described by the sequential composition of the three corresponding strategies.

Acknowledgment. We are grateful to Alberto Verdejo for his useful remarks regarding our implementation in Maude strategy language.

References

1. O. Andrei, G. Ciobanu, D. Lucanu. Executable Specifications of the P Systems. In *Membrane Computing. WMC5*, Lecture Notes in Computer Science 3365, Springer, 127–146, 2005.
2. O. Andrei, G. Ciobanu, D. Lucanu. A Structural Operational Semantics of the P Systems, In *Membrane Computing. WMC6*, Lecture Notes in Computer Science 3850, Springer, 32–49, 2006.
3. O. Andrei, G. Ciobanu, D. Lucanu. *Strategies and Tactics in Operational Semantics*, "A.I.Cuza" University, Faculty of Computer Science Tech. Rep. TR06-01, 2006.
4. C. Bonchiş, G. Ciobanu, C. Izbaşa. Encodings and Arithmetic Operations in Membrane Computing. In *Theory and Applications of Models of Computation*, Lecture Notes in Computer Science 3959, Springer, 618–627, 2006.
5. P. Borovansky, C. Kirchner, H. Kirchner, C. Ringeissen. Rewriting with Strategies in ELAN: A Functional Semantics. *International Journal of Foundations of Computer Science*, 12(1), 69-95, 2001.
6. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, J.F. Quesada. Maude: Specification and Programming in Rewriting Logic. *Theoretical Computer Science*, 285(2), 187-243, 2002.
7. N. Mart-Oliet, J. Meseguer, A. Verdejo. Towards a Strategy Language for Maude. *Electr. Notes Theor. Comput. Sci.*, 117, 417–441, 2005
8. J. Meseguer. Conditional Rewriting Logic as Unified Model of Concurrency. *Theoretical Computer Science*, 96, 73–155, 1992.
9. Gh. Păun. *Membrane Computing. An Introduction*. Springer, 2002.
10. E. Visser. A Survey of Strategies in Rule-Based Program Transformation Systems. *Journal of Symbolic Computation*, 40, 831–873, 2005.
11. E. Visser, Z.-A. Benaissa, A. Tolmach. Building Program Optimizers with Rewriting Strategies. *ACM SIGPLAN Notices*, 34, 13–26, 1999.

Tissue P Systems with Communication Modes

Francesco Bernardini[1,⋆] and Rudolf Freund[2]

[1] Leiden Institute of Advanced Computer Science
Leiden University
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
bernardi@liacs.nl
[2] Faculty of Informatics
Vienna University of Technology
Favoritenstr. 9, A-1040 Wien, Austria
rudi@emcc.at

Abstract. The paper introduces communication modes in tissue P systems that are based on the applicability of the rules to the objects present inside the cells. This notion of a communication mode is inspired by the concept of a derivation mode used in the area of grammar systems. Three different communication modes are identified depending on both the way the objects are moved from one cell to another one, altogether as a multiset or independently from each other, and on the moment when communication can take place, immediately after a terminal object is produced inside a cell, immediately after a cell has reached a terminal configuration or only when the system as a whole has reached a final configuration. The computational power of tissue P systems with different communication modes is compared with the power of the basic model of P systems and some classes of L systems.

1 Introduction

P systems represent a class of distributed and parallel computing devices of a biological type that was introduced in [12]. Several variants of this model have been investigated and the literature on the subject is still rapidly growing. The main results in this area show that P systems are a very powerful and efficient computational model (e.g., see [12], [13] and [17] for a comprehensive bibliography). The main ingredient of a P system is the membrane structure defined as a hierarchical arrangement of different membranes, embedded in a unique skin membrane, that identify several distinct regions inside the system. Each region gets assigned a finite multiset of objects and a finite set of rules for modifying the objects or moving them from one place to another one. The structure of a P system is usually represented as a tree. Thus, tissue P systems were proposed as a variant of membrane systems where the structure of the system is defined as an arbitrary graph [13], and which are somehow inspired by cell behaviour in

⋆ Research supported by NWO, the Netherlands Organisation for Scientific Research, project 635.100.006 "VIEWS".

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 170–182, 2006.

tissues of multi-cellular organisms (e.g., see [1]). Specifically, nodes in the graph represent cells that are able to communicate objects along the edges of the graph. The few variants of tissue P systems considered in the literature essentially differ in the mechanisms used to communicate objects from one cell to another one. For instance, particular sets of communication rules can be assigned to the edges in the graph defining the structure of the system in order to model the existence of communication channels among the cells in the systems [11], [9]; alternatively, there are evolution-communication tissue P systems, so to adopt the terminology introduced in [4], where the objects produced by particular transformations occurring inside the cells are non-deterministically propagated from one place to another one [10], [3].

In this paper, we consider a model for tissue P systems where each cell gets assigned a finite set of transformation rules (i.e., multiset rewriting rules) and, after each application of these rules, all the objects produced inside the cell that can be communicated are moved out from the cell and distributed to the neighbouring cells according to a specific communication mode. Specifically we identify three communication modes: *terminal at the level of objects*, *terminal at the level of cells*, and *terminal at the level of the system*. Terminal at the level of objects means that every object that cannot undergo any transformation inside a cell is communicated independently from the others immediately after it has been produced inside that cell. Terminal at the level of cells means that communication in a cell can only take place when no more transformation rules can be used inside the cell; the objects associated with that cell are then moved altogether at the same time and they are bound to reach the same target cell. Terminal at the level of the system means that no communication is permitted in the system till a configuration is reached where no more rules can be used inside any cell of the system; cells can then exchange their respective multisets of objects with the constraint that objects previously associated with the same cell must be moved altogether into the same target cell. In all these communication modes, by target cell or neighbouring cell, we mean a cell that is adjacent in the underlying graph to the cell where objects are moved from.

This notion of communication modes is inspired by the concept of derivation modes used in the area of grammar systems [5]. Grammar systems are systems formed by a number of grammar components that cooperate according to a given protocol in order to rewrite a common string. In the basic model of grammar systems, called CD grammar systems, a component at a time is non-deterministically chosen to rewrite the unique string in the system and, once activated, this component performs as many derivation steps as necessary according to the derivation mode chosen – $= k$, $\leq k$, $\geq k$ derivation steps, an arbitrary number of derivation steps or as many derivation steps as possible (t-mode). The common string is then released so that another component can become active etc. till a string accepted by the system is produced. Here, in some sense, we are reversing the perspective by considering the objects as the active elements in the system that are moved from one place to another one;

specifically, once some objects have entered a cell, they have to remain inside that cell until a specific condition is met such that a communication can take place and the objects can leave the cell. Moreover, with respect to the basic notion of cooperating/distributed grammar systems, different cells can be active at the same time by operating in parallel on different multisets of objects; as well as this, the use of multisets allows every object to evolve independently from the others and to be arbitrarily distributed to the cells in the system. We also note that similar issues concerning relationships between P systems and grammar systems were investigated in [2], [6] [8], [14].

The present paper is organised as it follows. Section 2 briefly recalls the notations commonly used in membrane computing and the few notions of formal language theory that will be used in the rest of the paper; in particular, we report the definition of extended tabled Lindenmayer systems including the case of systems with random contexts. Section 3 is dedicated to the definition of tissue P systems with a communication mode in the three cases mentioned above. The computational power of these classes of tissue P systems is then investigated in Section 4 in comparison with the power of the basic model of P systems and the power of extended tabled Lindenmayer systems. Some further remarks and directions for future research are discussed in the last section of the paper.

2 Preliminaries

We here recall some basic notions concerning the notations commonly used in membrane computing and the few notions of formal language theory we need in the rest of the paper. We refer to [13], [16] for further details.

An alphabet is a finite non-empty set of abstract symbols. Given an alphabet V, by V^* we denote the set of all possible strings over V, including the empty string λ. The length of a string $x \in V^*$ is denoted by $|x|$ and, for each $a \in V$, $|x|_a$ denotes the number of occurrences of the symbol a in x. A multiset over V is a mapping $M : V \longrightarrow \mathbb{N}$ such that $M(a)$ defines the multiplicity of a in the multiset M ($\mathbb{N}$ denotes the set of non-negative integers). Such a multiset can be represented by a string $a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)} \in V^*$ and by all its permutations, with $a_j \in V$, $M(a_j) \neq 0$, $1 \leq j \leq n$. In other words, we can say that each string $x \in V^*$ identifies a finite multiset over V defined by $M_x = \{ (a, |x|_a) \mid a \in V \}$. Given two multisets x and y, with $x, y \in V^*$, we say that the multiset x includes the multiset y, or the multiset y is included in the multiset x, and we write $x \sqsupseteq y$, or $y \sqsubseteq x$, if and only if $|x|_a \geq |y|_a$, for every $a \in V$. In P systems, in order to manipulate multisets, we need the notion of multiset rewriting rules as counterpart of the well-known operation of string rewriting. Formally, a multiset rewriting rule is a pair (x, y) – we also write $x \rightarrow y$ – with x, y being two finite multisets over V. A finite multiset w over V can be rewritten by a rule $x \rightarrow y$ if and only if $w \sqsupseteq x$, and we say that the multiset w can evolve by means of the rule $x \rightarrow y$ thereby replacing the multiset x by the multiset y in w, i.e. the result of the application of the rule $x \rightarrow y$ to the

multiset x is the multiset w' such that, for every $a \in V$, $|w'|_a = |w|_a - |x|_a + |y|_a$. Moreover, a multiset rewriting rule $x \to y$ is said to be non-cooperative if $|x| = 1$ whereas it is said to be cooperative if $|x| > 1$.

An ET0L system is a construct $G = (V, T, w, P_1, \ldots, P_m)$, $m \geq 1$, where V is an alphabet, $T \subseteq V$ is the terminal alphabet, $w \in V^*$ is the *axiom*, and P_i, $1 \leq i \leq m$, are finite sets of rules (*tables*) of non-cooperative rules over V of the form $a \to x$. In a derivation step, all the symbols present in the current sentential form are rewritten using one table. The language generated by G, denoted by $L(G)$, consists of all the strings over T which can be generated in this way by starting from w. An ET0L system with only one table is called an E0L system. By $E0L$ and $ET0L$ we denote the families of languages generated by E0L systems and ET0L systems, respectively. An ET0L system *with random contexts* [7] is an ET0L system where each table P is associated with a corresponding finite subset Q of V and the table P is applicable to a a given string if and only if this string contains each of the letters in Q. The family of languages generated by ET0L systems with random contexts is denoted by $ET0L(rc)$. It is known from [16] that $CF \subset E0L \subset ET0L \subset CS$, with CF being the family of context-free languages and CS being the family of context-sensitive languages; it is also proved in [7] that $ET0L \subset ET0L(rc)$. However, as the paper deals with P systems with symbol objects, we will consider ET0L systems as devices that generate sets of non-negative integers; to this aim, given an ET0L system G, we define the set of non-negative integers generated by G as the length set $N(G) = \{\, |x| \mid x \in L(G) \,\}$. The corresponding family of sets of non-negative integers then are denoted by NCF, $NE0L$, $NET0L$, $NET0L(rc)$, and NCS, respectively. Finally, we recall the fact that, according to Theorem 1.3 in [15], for each language $L \in ET0L$ there is an ET0L system that generates L, contains only two tables, i.e., $G = (V, T, w, P_1, P_2)$, and moreover, after having used P_1, we can use any of P_1 and P_2, but, after having used P_2, we always have to use P_1.

3 Tissue P Systems with a Communication Mode

Now we formally introduce the notion of tissue P systems with a communication mode by giving the following definition.

Definition 1. *A* tissue P system with a communication mode *is a construct*

$$\mathcal{T} = (V, \gamma, C_1, C_2, \ldots, C_n, c_O, \sigma)$$

where

1. *V is a finite alphabet of symbols called* objects*;*
2. *$\gamma = (\{1, 2, \ldots n\}, E)$, with $E \subseteq \{\{i, j\} \mid 1 \leq i, j \leq n\,,\ i \neq j\}$, is a finite connected undirected graph;*
3. *$C_i = (w_i, R_i)$, for each $1 \leq i \leq n$, such that*
   (a) $w_i \in V^*$ is a finite multiset of objects;

(b) *R_i is a finite set of multiset rewriting rules of the form $a \to y$ for $a \in V$ and $y \in V^*$; these rules are called the* transformation rules *associated with cell i;*

4. *c_O is the* output cell*;*
5. *$\sigma \in \{tObj, tCell, tSys\}$ specifies the communication mode adopted by the system.*

A tissue P system with a communication mode, which, for the sake of simplicity, in the following will be just called a tissue P system, is defined as a collection of $n \geq 1$ cells that are associated in a one-to-one manner to the nodes of a finite undirected graph denoted by γ that are labeled by values in $\{1, 2, \ldots, n\}$. The edges of the graph γ define the existing links between the cells of the system, and they are represented as unordered pairs of the form $\{i, j\}$ with $1 \leq i, j \leq n$ and $i \neq j$. Each cell C_i, with $i \in \{1, 2, \ldots, n\}$, represents a basic functional unit of a tissue P system and it is characterized by a finite multiset of objects w_i, which defines its initial contents, and by a finite set of multiset rewriting rules R_i, called transformation rules, which allow a cell to modify its contents by consuming some objects in order to produce some new ones. Throughout the rest of the paper, each cell C_i, with $i \in \{1, 2, \ldots, n\}$, will be referred to as cell i. Moreover, for each cell i, we define the set of its neighbouring cells $\mathcal{N}_i = \{j \mid \{i, j\} \in E\}$ (i.e., the set of cells that are directly linked to cell i according to the graph γ) and its terminal alphabet $T_i = \{a \mid$ there exists no rule $a \to v \in R_i$, for any $v \in V^*\}$ (i.e., the set of objects that cannot evolve by means of any rule in R_i).

Communication in a tissue P system $\mathcal{T}$ is driven by the communication mode $\sigma \in \{tObj, tCell, tSys\}$, which specifies how objects can be moved from one cell to another one; specifically, for each cell i, with $1 \leq i \leq n$, communication involves only the objects in T_i and it can be done in the following ways:

- *tObj* (*terminal at the level of objects*): every time an object $a \in T_i$ is produced inside cell i, it is immediately moved from cell i to a cell $j \in \mathcal{N}_i$ non-deterministically chosen; the objects associated with cell c_O (the output one), i.e., the objects from T_{c_O}, can never be moved out from cell c_O.
- *tCell* (*terminal at the level of cells*): communication of objects from cell i to another one takes place only when all the objects contained in cell i are in T_i (i.e., when the objects inside cell i cannot evolve anymore by means of any rule in R_i); the objects in T_i contained in cell i are then moved altogether at the same time from cell i to the same cell $j \in \mathcal{N}_i$ non-deterministically chosen; the objects from T_{c_O} can never be moved out from cell c_O.
- *tSys* (*terminal at the level of the system*): in a way similar to the communication mode *tCell*, the objects in T_i contained in cell i are moved altogether at the same time from cell i to the same cell $j \in \mathcal{N}_i$ non-deterministically chosen; however, before any communication can take place, the system must have reached a configuration where no more transformation rules can be used inside any cell of the system; the objects from T_{c_O} can never be moved out from cell c_O.

Thus, a computation in a tissue P system $\mathcal{T}$ consists of a sequence of steps where, after each application of the transformation rules, all the communications permitted by the communication mode σ take place and the new distribution of objects inside the system becomes effective when starting the next step of computation. As is quite usual in P systems, transformation rules are applied in a non-deterministic maximally parallel manner. A computation in $\mathcal{T}$ is said to be *successful* if it reaches a configuration where no more transformation rules can be used inside any cell of the system and no further communications between the cells can take place. The result of a successful computation is given by the number of objects contained in the output cell c_O in the final configuration. The set of non-negative integers generated by all the successful computations in $\mathcal{T}$ is denoted by $N(\mathcal{T})$. Notice that, whatever the communication mode is, a successful computation always produces a final configuration where all the cells but the output one are empty, as $\gamma = (\{1, 2, .., n\}, E)$ is a connected graph, i.e., for each pair (i, j), $1 \leq i, j \leq n$ and $i \neq j$, it is always possible to reach node j from node i through a path in the graph γ.

4 The Computational Power of Tissue P Systems with a Communication Mode

In this section we present some results concerning the generative power of tissue P systems when different communication modes are used. To this aim, we introduce the families of sets of non-negative integers of the form $NOTP_n(\sigma)$, with $n \geq 1$ and $\sigma \in \{tObj, tCell, tSys\}$; they identify the families of sets of non-negative integers generated by tissue P systems with at most n cells that adopt the communication mode σ. Moreover, when the number of cells is not specified but it can assume any value, the corresponding family of sets of non-negative integers is denoted by $NOTP_*(\sigma)$, with σ as above.

The first result shows that, in the case $tObj$, the hierarchy on the number of membranes collapses at level 1, and this communication mode in fact does not increase the power of (tissue) P systems with respect to the basic model of membrane systems.

Theorem 1. *$NOTP_*(tObj) = NOTP_n(tObj) = NCF$, for $n \geq 1$.*

Proof. Let $\mathcal{T} = (V, \gamma, (w_1, R_1), (w_2, R_2), \ldots, (w_n, R_n), c_O, tObj)$ be a tissue P system as specified in Definition 1, with $n \geq 1$. For each $1 \leq i \leq n$ and $a \in V$, we define the set of possible destinations for the object a that can be reached from cell i as the set $D_i(a)$ such that

- $D_i(a) = \{ a_j \mid j \in \mathcal{N}_i \}$, if $c_O \notin \mathcal{N}_i$ or $a \notin T_{c_O}$, and
- $D_i(a) = \{ a_j \mid j \in \mathcal{N}_i, j \neq c_O \} \cup \{a\}$, if $c_O \in \mathcal{N}_i$ and $a \in T_{c_O}$.

Then we construct a tissue P systems $\mathcal{T}'$ with one cell that is able to simulate $\mathcal{T}$:

$$\mathcal{T} = (V', (\{1\}, \emptyset), (w', R'), 1, tObj)$$

where $V' = T_{c_O} \cup \{ a_i \mid a \in V, 1 \leq i \leq n\}$, $w' = h_1(w_1)h_2(w_2)\dots h_n(w_n)$, for $h_i(a) = a_i$, if $i \neq c_O$ or $a \notin T_{c_O}$, $h_i(a) = a$, if $i = c_O$ and $a \in T_{c_O}$, $1 \leq i \leq n$, and R' is a finite set of rules that contains

- for each rule $a \to b_1 b_2 \dots b_k \in R_i$, with $k \geq 1$, $b_1, b_2, \dots, b_k \in V$, $i \neq c_O$, a finite set of rules $R'_i(a \to b_1 b_2 \dots b_k)$ such that

$$R'_i(a \to b_1 b_2 \dots b_k) = \{ a_i \to b'_1 b'_2 \dots b'_k \mid b'_j = h_i(b_j) \text{ if } b_j \notin T_i \text{ and } b'_j \in D_i(b_j) \text{ if } b_j \in T_i, 1 \leq j \leq k \};$$

- for each rule $a \to v \in R_{c_O}$, with $v \in V^*$, a rule $a \to h_{c_O}(v)$;
- for each $a \in T_i$, with $i \neq \{c_O\}$, a set of rules

$$R'_i(a) = \{ a_i \to b \mid b \in D_i(a)\};$$

- for each rule $a \to \lambda \in R_i$, with $1 \leq i \leq n$, a rule $a_i \to \lambda$.

The simulation of the tissue P system $\mathcal{T}$ by means of the tissue P system $\mathcal{T}'$ is done in the following way: each rule $a \to b_1 b_2 \dots b_k \in R_i$, with $k \geq 1$, $b_1, b_2, \dots, b_k \in V$, $1 \leq i \leq n$, $i \neq c_O$, is mapped to a set of rules $R'_i(a \to b_1 b_2 \dots b_k)$; these rules simulate the application of the rule $a \to b_1 b_2 \dots b_k \in R_i$ to the objects contained in cell i and, at the same time, they simulate the communication of the objects in T_i that appear on the right-hand side of the rule. This is done by replacing, on the right-hand side of the rule, each object $b \notin T_i$ by an object b_i (i.e., the object b remains inside cell i) and each object $b \in T_i$ by an object $c \in D_i(a)$ (i.e., the object b is moved from cell i to one of the neighbouring cells). Notice that we have to consider all the possible combinations for the destinations of the objects because communication in the tissue P system $\mathcal{T}$ is non-deterministic and, as the communication *tObj* is adopted, each object is moved independently from the others. In other words, the non-determinism at the level of communication observed in the tissue P system $\mathcal{T}$ is reflected by the non-determinism at the level of the transformation rules associated with the unique cell of the tissue P system $\mathcal{T}'$. The simulation of the application of the rules inside the output cell c_O is considered apart because the objects can never be moved out from that cell. Specifically, for each rule $a \to v \in R_{c_O}$, there exists a corresponding rule $a \to h_{c_O}(v)$ in $\mathcal{T}'$ such that all the objects get assigned the label c_O. Finally, each object $a \in T_i$ that enters a cell i is immediately moved out from that cell by using a rule $a_i \to b$, with $b \in D_i(a)$.

Therefore we infer that the tissue P system $\mathcal{T}'$ correctly simulates the applications of the rules in $\mathcal{T}$ and, in the case of a successful computation in $\mathcal{T}'$, the multiset of objects obtained inside the unique cell in $\mathcal{T}'$ is exactly the same as the multiset produced inside the output cell c_O by the corresponding

successful computation in $\mathcal{T}$. Therefore $N(\mathcal{T}') = N(\mathcal{T})$, and thus we have proved $NOTP_*(tObj) = NOTP_n(tObj)$, for $n \geq 1$.

Then, in order to show that $NOTP_1(tObj) = NCF$, we can use a similar result as proved in [13] for the basic model of P systems; in [13], a characterisation of NCF is provided that is based on P systems with one membrane using only non-cooperative rules. Thus, such a result can be immediately transferred to tissue P systems with one cell as it is obvious that the two models are equivalent in the case of systems with only one cell. □

The communication mode $tSys$ can instead be proved to be more powerful than the communication mode $tObj$:

Theorem 2. *$NOTP_*(tSys) = NOTP_n(tSys) = NET0L$, for $n \geq 4$.*

Proof. **(i)** $NET0L \subseteq NOTP_4(tSys)$. As pointed out in Section 2, for every language $L \in ET0L$ there exists an ET0L system G with only two tables that generates L, i.e., $G = (V, T, w, P_1, P_2)$. Moreover, after having used table P_1, we can use both P_1 and P_2, but, after having used table P_2, we always have to use table P_1. The table used in the first step of a computation is P_1. In order to simulate G, we construct the following tissue P system

$$\mathcal{T}_L = (V', \gamma, (w', R_1), (\lambda, R_2), (\lambda, R_3), (\lambda, R_4), 4, tSys)$$

where

$$\begin{aligned}
V &= \{a, a', a'' \mid a \in V\} \cup \{\#\},\\
\gamma &= (\{1,2,3,4\}, \{\{1,2\}, \{1,3\}, \{1,4\}, \{2,4\}\}),\\
R_1 &= \{\, a' \to b_1'' b_2'' \dots b_k'' \mid a \to b_1 b_2 \dots b_k \in P_1, k \geq 1 \,\}\\
&\quad \cup \{\, a' \to \lambda \mid a \to \lambda \in P_1 \,\},\\
R_2 &= \{\, a'' \to b_1' b_2' \dots b_k' \mid a \to b_1 b_2 \dots b_k \in P_2, k \geq 1 \,\}\\
&\quad \cup \{\, a'' \to \lambda \mid a \to \lambda \in P_2 \,\},\\
R_3 &= \{\, a'' \to a' \mid a \in V \,\},\\
R_4 &= \{\, a' \to a, a'' \to a \mid a \in T \,\}\\
&\quad \cup \{\, a' \to \#, a'' \to \# \mid a \in (V - T) \,\} \cup \{\, \# \to \# \,\}.
\end{aligned}$$

The simulation of the ET0L system G by means of the tissue P system $\mathcal{T}$ is done in the following way.

Cell 1 is used to simulate table P_1: the application of rules in P_1 produces a multiset $u'' \in \{\, a'' \mid a \in V\}^*$ inside cell 1; after that, no more rules can be used in the system and u'' is moved from cell 1 either to cell 2 or to cell 3 or to cell 4. If the multiset u'' reaches cell 2, then we can pass to simulate the application of P_2. If the multiset u'' instead reaches cell 3, then we return to simulate P_1; this is done by replacing each object a'' by the object a' and moving all these objects back to cell 1. In cell 2, the simulation of table P_2 is done in a similar way by applying corresponding transformation rules that produce a multiset

$u' \in \{ a' \mid a \in V \}^*$; after that, no more rules can be used in the system and u' is moved from cell 2 either to cell 1, to simulate another application of table P_1, or to cell 4. At any moment, if a multiset $u' \in \{ a' \mid a \in V \}^*$ or a multiset $u'' \in \{ a'' \mid a \in V \}^*$ reaches cell 4, then the rules in R_4 are used to check whether this multiset contains only terminal symbols or not. Specifically, every object a' or a'', with $a \in T$, is replaced by the corresponding terminal object a, whereas every object a' or a'', with $a \in (V - T)$, is replaced by the object #. Thus, if the multiset u' or the multiset u'' contains some non-terminal symbol, an infinite computation is the consequence; otherwise, the computation halts.

Therefore, the tissue P system $\mathcal{T}_L$ correctly simulates the ET0L system G and we conclude $N(G) = N(\mathcal{T}_L)$.

(ii) $NOTP_n(tSys) \subseteq NET0L$, for $n \geq 1$. Consider a tissue P system $\mathcal{T}$ as specified in Definition 1 such that

$$\mathcal{T} = (V, \gamma, (w_1, R_1), (w_2, R_2), \ldots, (w_n, R_n), c_O, tSys),$$

with $\gamma = (\{1, 2, \ldots, n\}, E)$, $n \geq 1$.Then, let $\gamma' = (\{1, 2, \ldots, n\}, E')$ be a directed graph such that, for all $\{i, j\} \in E$, $(i, j) \in E$ and $(j, i) \in E$, and nothing else is in E'. We construct the following ET0L system that simulates the behaviour of the tissue P system $\mathcal{T}$:

$$G = (V', V, w, \mathcal{P})$$

where $V' = \{ a_i \mid a \in V, 1 \leq i \leq n \} \cup V \cup \{\#\}$, $w = h_1(w_1)h_2(w_2) \ldots h_n(w_n)$, with $h_i(a) = a_i$, $1 \leq i \leq n$, $a \in V$, and $\mathcal{P}$ is a finite set of tables that contains

- a table

$$\begin{aligned} P_1 = & \{ a_i \to h_i(v) \mid a \to v \in R_i, a \in V, v \in V^*, 1 \leq i \leq n \} \\ & \cup \{ a_i \to a_i \mid a \in T_i, 1 \leq i \leq n \} \\ & \cup \{ a \to a \mid a \in V \} \cup \{\# \to \#\}; \end{aligned}$$

  this table is used to simulate the application of transformation rules for the objects currently associated with the cells in the system; these rules must be applied as many times as possible until a configuration is reached where no more transformation rules can be used in the system;
- a table

$$\begin{aligned} P_{\{(i_1,j_1),(i_2,j_2),\ldots,(i_{n-1},j_{n-1})\}} = & \{ a_{i_h} \to a_{j_h} \mid a \in T_{i_h}, 1 \leq h \leq n-1 \} \\ & \cup \{ a_i \to \# \mid a \notin T_i, 1 \leq i \leq n \} \cup \{\# \to \#\} \\ & \cup \{ a_{c_O} \to a_{c_O} \mid a \in T_{c_O} \} \\ & \cup \{ a \to a \mid a \in V \}, \end{aligned}$$

  for each $\{(i_1, j_1), (i_2, j_2), \ldots, (i_{n-1}, j_{n-1})\} \subseteq E'$, with

$$\{i_1, i_2, \ldots, i_{n-1}\} = \{ i \mid 1 \leq i \leq n, i \neq c_O \};$$

  these tables are used to simulate the communication between the cells in the system; specifically, for each h, $1 \leq h \leq n - 1$, we choose a target cell

$j_h \in \mathcal{N}_{i_h}$ (the objects in T_{c_O} can never be moved out from cell c_O), and the communication of objects from cell i_h to cell j_h then is performed by just changing the label of the objects from i_h to j_h; moreover, rules of the form $a_i \to \#$, with $1 \leq i \leq n$, $a_i \notin T_i$, are considered in order to make sure that these tables are used only at the right moment when the objects inside the cells cannot evolve anymore by means of any rule; also notice that, since, for each edge $\{i, j\} \in E$, the set E' contains both (i, j) and (j, i), we have all the tables necessary to simulate communication between two cells in any direction;

– a table

$$\begin{aligned} P_f = &\{ a_i \to \# \mid a \in V, 1 \leq i \leq n,\ i \neq c_O \} \\ &\cup \{ a_{c_O} \to \# \mid a \notin T_{c_O} \} \\ &\cup \{ a_{c_O} \to a \mid a \in T_{c_O} \} \\ &\cup \{ a \to a \mid a \in V \} \cup \{\# \to \#\}; \end{aligned}$$

this table is used to finish a successful computation in $\mathcal{T}$; in fact, as pointed out in Section 3, a computation in a tissue P system can be successful if and only if the system reaches a configuration where all the cells except for the output one are empty and the output cell contains only objects that cannot evolve anymore by means of any rule associated with that cell.

Therefore, we conclude that the L system G correctly simulates the tissue P system $\mathcal{T}$, and we have $N(G) = N(\mathcal{T})$. □

Finally, we pass to consider the communication mode $tCell$. As an immediate consequence of Theorem 2, we obtain the following result.

Corollary 1. *$NET0L \subseteq NOTP_n(tCell)$, for $n \geq 4$.*

It is in fact easy to see that the behaviour of the tissue P system $\mathcal{T}_L$ as defined in the proof of Theorem 2, would not be changed if the communication mode $tCell$ were adopted instead of the communication mode $tSys$, because only one cell at a time is active in $\mathcal{T}_L$ and, once the application of its own rules has been finished, the resulting multiset is passed to another cell, which then becomes the new active one, etc. On the other hand, for the communication mode $tCell$, we are not able to prove the opposite inclusion or to prove that the hierarchy on the number of membrane collapses at a particular level. Nevertheless, we can provide an upper bound for the generative capacity of this class of tissue P systems by considering ET0L systems with random contexts:

Lemma 1. *$NOTP_*(tCell) \subseteq NET0L(rc)$.*

Proof. We adopt a construction very similar to the one used in point **(ii)** of the proof of Theorem 2. Therefore, we do not report all the details of the proof and leave the task of a complete formalisation to the reader.

Consider a tissue P system $\mathcal{T}$ as specified in Definition 1 such that

$$\mathcal{T} = (V, \gamma, (w_1, R_1), (w_2, R_2), \ldots, (w_n, R_n), c_O, tSys),$$

with $\gamma = (\{1, 2, \ldots, n\}, E)$, $n \geq 1$.Then, let $\gamma' = (\{1, 2, \ldots, n\}, E')$ be a directed graph such that, for all $\{i, j\} \in E$, $(i, j) \in E$ and $(j, i) \in E$, and nothing else is in E'. We construct an ET0L system with random contexts G that simulates the tissue P system $\mathcal{T}$ as follows: first of all, we define the table

$$\begin{aligned} P_1 = &\{ a_i \to h'_i(v) \mid a \to v \in R_i, 1 \leq i \leq n \} \\ &\cup \{ a_i \to a'_i \mid a \in T_i, 1 \leq i \leq n \} \\ &\cup \{ a'_i \to \# \mid a \in V, 1 \leq i \leq n \} \\ &\cup \{ a \to a \mid a \in V \} \cup \{ \# \to \# \}, \\ Q = &\emptyset, \end{aligned}$$

with $h'_i(a) = a'_i$, for each $a \in V$, $1 \leq i \leq n$. This table is used to simulate a single application of transformation rules to the objects currently contained inside the cells; for each $1 \leq i \leq n$, the objects currently assigned to cell i are of the form a_i, for some $a \in V$, and, after the application of the transformation rules, all these objects turn to be of the form a'_i. This is necessary because, after each application of the transformation rules, we have to check for the possibility that some communication can take place inside some cells that have reached a configuration where no more rules can be used. To this aim, we consider all the possible partitions $I = I_1 \cup I_2$, with $I_1 \cap I_2 = \emptyset$, of the set $I = \{ i \mid 1 \leq i \leq n, i \neq c_O \}$, i.e., I_1 identifies the set of cells where communication can take place, whereas I_2 identifies the set of cells where no communication is permitted. For each partition of that kind, we construct all tables of the form

$$\begin{aligned} P_{\{(i_1,j_1),(i_2,j_2),\ldots,(i_k,j_k)\}} = &\{ a'_{i_h} \to a_{j_h} \mid a \in T_{i_h}, 1 \leq h \leq k \} \\ &\cup \{ a'_{i_h} \to \# \mid a \notin T_{i_h}, 1 \leq h \leq k \} \\ &\cup \{ a'_{p_t} \to a_{p_t} \mid a \in V, 1 \leq t \leq m \} \\ &\cup \{ a_i \to \# \mid a \in V, 1 \leq i \leq n \} \\ &\cup \{ a'_{c_O} \to a_{c_O} \mid a \in V \} \\ &\cup \{ a \to a \mid a \in V \} \cup \{ \# \to \# \}, \\ Q = &\{ b_{p_1}, b_{p_2}, \ldots, b_{p_m} \}, \end{aligned}$$

such that $\{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\} \subseteq E'$, $I_1 = \{i_1, i_2, \ldots, i_k\}$, and $b_{p_1} \notin T_{p_1}$, $b_{p_2} \notin T_{p_2}$,..., $b_{p_m} \notin T_{p_m}$, with $I_2 = \{p_1, p_2, \ldots, p_m\}$. These tables are necessary to simulate the communication of objects between the cells in the tissue P system $\mathcal{T}$. Given I_1 and I_2, the random context makes sure that one of these table is used if and only if each cell p, with $p \in I_2$, contains at least one object $b_p \notin T_p$ (i.e., no communication is permitted for any cell in I_2). On the other hand, communication of objects from a cell i, with $i \in I_1$, can take place if and only if cell i does not contain any object $a \notin T_i$; if this is not the

case, then the trap object # is introduced. Finally, after the application of one of these communication tables, we can either return to apply table P_1 or decide to finish the computation by applying the following final table:

$$\begin{aligned} P_f = &\{ a'_i \to \# \mid a \in V, 1 \leq i \leq n, i \neq c_O \} \\ &\cup \{ a_i \to \# \mid a \in V, 1 \leq i \leq n \} \\ &\cup \{ a'_{c_O} \to \# \mid a \notin T_{c_O} \} \\ &\cup \{ a'_{c_O} \to a \mid a \in T_{c_O} \} \\ &\cup \{ a \to a \mid a \in V \} \cup \{\# \to \#\}, \\ Q = &\emptyset. \end{aligned}$$

This table is used to finish a successful computation in T; in fact, as pointed out in Section 3, a computation in a tissue P system can be successful if and only if the system reaches a configuration where all the cells except for the output one are empty and the output cell contains only objects that cannot evolve anymore by means of any rule associated with that cell.

Therefore, we conclude that the L system G correctly simulates the tissue P system $\mathcal{T}$, and we have $N(G) = N(\mathcal{T})$. □

5 Conclusions

Membrane computing and grammar systems are two active areas of theoretical computer science, with different starting points, but with several similarities (both areas deal with distributed computing devices, where such notions as parallelism, cooperation, decentralisation are crucial). P systems were introduced with a clear biological motivation by being mainly inspired by the structure and functioning of living cells [13]. The initial motivation of grammar systems instead was related to artificial intelligence issues, and only later on features from other areas of parallel and distributed computing were incorporated [5]. The present work attempts to bridge the areas of P systems and grammar systems by introducing a notion of a communication mode for tissue P systems that is inspired by the concept of a derivation mode used in grammar systems. The notion of a communication mode is mainly related to the t-mode of derivation: rewrite a string as many times as possible till a terminal string is produced [5]. In particular, the result proved in Theorem 2 is in a sense coherent with the similar characterisation of $ET0L$ provided in [5] for grammar systems with the t-mode of derivation. Future research in this respect may be dedicated to better characterize the concepts of communication mode, active component, terminal derivation in the context of P systems. Other investigations may be dedicated to study P systems with string objects in relationship to the two standard models of grammar systems: CD grammar systems and PC grammar systems. Approaches in this direction can already be found in [2], [6].

References

1. Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., Walter, P.: The Molecular Biology of the Cell. Fourth Edition. Garland Publ. Inc. London (2002)
2. Bernardini, F., Gheorghe, M.: Population P Systems and Grammar Systems. In: E. Csuhaj-Varjú, Gy. Vaszil (eds.): Proceedings of Grammar Systems Week 2004, Budapest, Hungary, July 5-9, 2004. MTA SZTAKI Budapest (2004) 66–77
3. Bernardini, F., Gheorghe, M.: Cell Communication in Tissue P Systems: Universality Results. Soft Computing **9**, 9 (2005) 640–649
4. Cavaliere, M.: Evolution-Communication P Systems. In: Păun, Gh., Rozenberg, G., Salomaa, A. and Zandron, C. (eds.): Membrane Computing. International Workshop, WMC-CdeA 02, Curtea de Argeş, Romania, August 19-23, 2002. Revised Papers. Lecture Notes in Computer Science **2597** Springer (2003) 134–145
5. Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, Gh.: Grammar Systems. A Grammatical Approach to Distribution and Cooperation. Gordon and Breach London (1994)
6. Csuhaj-Varjú, E., Păun, Gh., Vaszil, Gy.: Grammar Systems versus Membrane Computing: The Case of CD Grammar Systems. To appear in Fundamenta Informaticae (2006)
7. Dassow, J., Păun, Gh.: Regulated Rewriting in Formal Language Theory. Springer Berlin (1989)
8. Freund, R., Oswald, M.: Modelling Grammar Systems by Tissue P Systems Working in the Sequential Mode. In: E. Csuhaj-Varjú, Gy. Vaszil (eds.): Proceedings of Grammar Systems Week 2004, Budapest, Hungary, July 5-9, 2004. MTA SZTAKI Budapest (2004) 179–199
9. Freund, R., Păun, Gh., Pérez-Jiménez, M.J.: Tissue-like P systems with channel states. Theoretical Computer Science **330** (2005) 101–116
10. Martín-Vide, C., Păun, Gh., Pazos, J., Rodríguez-Patón, A.: Tissue P Systems. Theoretical Computer Science **296** (2003) 295–326
11. Păun, A., Păun, Gh., Rozenberg, G.: Computing by Communication in Networks of Membranes. International Journal of Foundations of Computer Science **13** (2002) 779–798
12. Păun, Gh.: Computing with Membranes. Journal of Computer and System Sciences **61** (2000) 108–143
13. Păun, Gh.: Membrane Computing. An Introduction. Springer (2002)
14. Păun, Gh.: Grammar Systems vs. Membrane Computing: A Preliminary Approach.In: E. Csuhaj-Varjú, Gy. Vaszil (eds.): Proceedings of Grammar Systems Week 2004, Budapest, Hungary, July 5-9, 2004. MTA SZTAKI Budapest (2004) 255–275
15. Rozenberg, G., Salomaa, A.: The Mathematical Theory of L Systems. Academic Press New York (1980)
16. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. 3 volumes. Springer (1997)
17. The P Systems Web Page. `http://psystems.disco.unimib.it`

Towards a Hybrid Metabolic Algorithm

Luca Bianco and Federico Fontana

University of Verona
Department of Computer Science
15 strada Le Grazie – 37134 Verona, Italy
bianco@sci.univr.it, federico.fontana@univr.it

Abstract. During recent years stochastic algorithms have deserved much attention from the computational biology research communities. In this paper we derive a hybrid version of the formerly known Metabolic Algorithm that is enriched with stochastic features, whose impact on the dynamics of the system is especially prominent when the amount of metabolite becomes smaller. This hybrid procedure represents a first attempt to let the Metabolic Algorithm deal with low concentrations of substances according to a non-deterministic policy.

1 Introduction

The simulation of a metabolic process relies on several, sometimes well-established methods and algorithms that either compute its evolution deterministically, as it happens with methods discretizing a Reaction Rate Equation, or stochastically, as it happens with algorithms solving or approximating the Chemical Master Equation [6,19]. The latter algorithms are quite accurate but computationally expensive, given the individual handling they must do of each molecule, and provided that only several repeated realisations of the same simulation in principle provide sufficient information about the expected behavior of a system. For this reason, approximated versions of these algorithms have been proposed in order to diminish the computational burden of the stochastic approach [4].

Less is known about the possibility to gain efficiency, without loosing too much in accuracy, by using *hybrid* algorithms capable of mixing the deterministic and the stochastic paradigms together. Such an approach is inherently hard to deal with due to the theoretical and technical difficulties that arise when a system, whose kinetic rates range among different scales, is split into a multiple observation-level model accounting for different computation strategies depending on the level of observation [19]. Nevertheless, such an approach has been already pursued for simulating complex biochemical systems [9,17] and also in the processing of proteomic data[1] [20].

Our work here contributes to this research area, but it is still far from adding substantial knowledge about this possibility. Despite this we will show how a hybrid strategy to biochemical system modeling can be implemented within

[1] An interesting introduction to the use of hybrid algorithms in computational biology can be found online at *http://www.bioinfo.de/isb/2004/04/0024/main.html*.

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 183–196, 2006.

MP systems, by extending their deterministic evolution mechanism in order to include randomness that is always present in biochemical systems.

Metabolic P (MP) systems [11,3] are rooted in the theory and formalism of P systems [13,14], to which they couple a deterministic computational strategy called P Metabolic (MP) Algorithm [12,1,3]. P systems have been envisioned to find solutions for several problems [5]. In the meantime they have been a fertile ground for the birth of stochastic algorithms for the representation of the evolution of biochemical systems.

The P system community has approached the question of stochastic evolution in several ways. One strategy is known as Dynamical Probabilistic P systems, and employs maximal parallelism both at the rule and object level: there, the probability of tossing a rule is dynamically calculated by starting from the multiset of objects that are present in the system during a transition, as well as from the kinetic constant associated to the rule. Another is called Multi-compartmental Gillespie's algorithm and its aim is to extend the Gillespie algorithm to a multi-compartment environment as it happens for P systems having more than one membrane [6,16,15].

By exploiting the versatility of the MP algorithm, we can straightforwardly integrate a stochastic strategy for choosing the strength of the rules governing the system evolution, in a way that the smaller the amount of a substance is, the stronger the effects of randomness. In practice this is made by altering the deterministic character of the reaction maps proportionally to their magnitude [3]. In the end of the paper this hybrid algorithm is tested upon traditional case studies such as the BZ reaction and the Lotka-Volterra dynamics [7,3,10,2].

In the following of the paper we assume the reader to be friendly enough with the notation and the formalism of MP systems, whose leading ideas are that:

- the system evolves by allocating to each evolutionary rule object amounts that play the role of reactants, as well as by obtaining from such rules object amounts that play the role of products;
- nonlinear functions of the state of the system (that is identified by the amount of every object), called *reaction maps*, are computed at the beginning of each transition for assigning object amounts to the rules.

For further details on MP systems and the MP algorithm (shortly MPA) we refer the reader to [11,3].

In this paper, after introducing the hybrid formulation of the MP algorithm, called *h-MPA*, simulation examples are provided of two versions of the Belousov-Zhabotinsky (BZ) reaction as well as the Lotka-Volterra dynamics.

2 The Algorithm

The idea of the *hybrid algorithm* is to switch from a completely deterministic (MPA) to a completely stochastic approach (s-MPA) depending on the size of the population dealt with by each rule. A threshold τ is used to control the switch between the two strategies and in this way the whole system becomes

a stochastic-deterministic hybrid system (h-MPA). If the population is small compared to the threshold, the stochastic strategy should be preferred, otherwise the deterministic strategy is able to provide an acceptable approximation of the dynamics and is thereby preferable.

In principle, for each rule a deterministic or stochastic strategy has to be chosen according to the size of the population it deals with. Let us suppose to have a rule $r : XY \to ...$, and to fix a threshold τ. If

$$\min(q(X), q(Y)) < \tau,$$

then the strategy of application of r is chosen to be stochastic, else it is deterministic—$q(Z)$ denotes the amount of the species Z present into the system. Note that this minimum gives the bottleneck of the reaction, but it is different from the limiter of the standard metabolic algorithm because here we do not take into account the strength of the rules. A population, then, undergoes a stochastic dynamics if its size is smaller than the threshold.

Of course, due to cooperation, a population can undergo a stochastic dynamics even if it is bigger than the threshold, but it is involved in reactions dealing with at least one reactant whose total amount in the system is below the threshold.

What do we mean by stochastic strategy? The idea is to keep a "population perspective" of the dynamics, as in the spirit of the metabolic algorithm. Accordingly, a stochastic strategy for the simulation of a rule r with the system in state $\mathbf{s}$ consists in:

i) evaluating the reaction map $F_r(\mathbf{s})$ as in the deterministic MPA;
ii) picking up a random number from a probability distribution depending on $F_r(\mathbf{s})$, let us call this number v;
iii) applying the rule as in MPA by using v instead of $F_r(\mathbf{s})$ as a reaction map,

where a generic state $\mathbf{s}$ can be thought as a vector of concentrations of all the elements of the system (that are assumed ordered) and it is denoted as $s\text{-}MPA(\mathbf{s})$ (we denote with $MPA(\mathbf{s})$ the purely deterministic evolution of state $\mathbf{s}$).

This pseudo-algorithm gives an intuitive idea of the process, but it does not simulate properly the application of rule r (for example, we need to specify stochastic reaction maps of all reactants of rule r). In Subsection 2.3 a full description of the hybrid algorithm is given. Before entering the description of the algorithm, another preliminary question need to be addressed.

How do we quantify the dependency of the probability distribution on F_r? The idea is to respect somehow the shape of the (deterministic) reaction map in the random choice of the stochastic reaction map. This is because reaction maps should take into account the features of the interactions between elements of the system and with this respect it seems reasonable to consider them as "independent of the scale" and thereby valid also for small populations of objects. Nevertheless, the generality of the approach allows the modeler to specify a different shape for the reaction maps employed in the stochastic part of the algorithm, but since this is not limiting, here we will not exploit this capability.

One possible implementation of the random step (ii) is to generate a random number v by using a pseudo random sequence generator (PRSG) with a gaussian

distribution of mean $F_r(\mathbf{s})$ and a suitable variance, allowing a certain degree of variability in the dynamics.

2.1 PRSG

Standard Matlab (but not only it) provides a primitive for reckoning a (pseudo) random number chosen from a normal distribution with mean zero and variance one. If we denote with v_{nor} such a random number when obtained from a normal distribution, then

$$v_{rnd} = m + \sigma \cdot v_{nor}$$

is a (pseudo)random number chosen from a gaussian distribution with mean m and variance σ^2.

This expression does not necessarily produce positive values. As reaction maps cannot assume negative values, our PRSG skips eventual negative values. This strategy introduces a distortion of the gaussian paradigm, and in the future we will look for a more coherent random generation of numbers.

In all experiments we have used the PRSG to obtain a stochastic reaction map by starting from a deterministic one F_r evaluated in a certain configuration $\mathbf{s}$ of the system, thereby we have used $m = F_r(\mathbf{s})$ as mean value and (see two histograms of random sequences of 100000 numbers depicted in Figure 1) with $\sigma^2 = 0.5 \cdot F_r(\mathbf{s})^2$ as variance.

Figure 1 suggests that $\sigma^2 = 0.5 \cdot F_r(\mathbf{s})^2$ is a possible choice for the variance, giving enough variability in the distribution of the random number.

2.2 Deterministic, Stochastic or Both?

A sudden switch from the deterministic strategy (when the algorithm deals with populations larger than the threshold) to the stochastic one seems to be unrealistic. For instance, according to Schrödinger, a system's tendency toward a random behavior is inversely proportional to the square root of the number of molecules [18].

Rather, our strategy configures itself as a continuous switch between the two regimes along various intermediate degrees of determinism. The idea in this case is to use a sigmoid function, whose input is the threshold and a population value and whose output is the degree of determinism of the system (i.e., a real value d in the unitary interval $[0, 1]$ determining the rate of change reckoned by means of the deterministic algorithm). Of course, the value $1 - d$ is the degree of stochasticity of the system. Given a threshold τ and an input value x specifying the population size of the species considered, the value of the determinism degree d produced by the sigmoid function can be computed as

$$d = \frac{1}{1 + e^{(10/\tau)(\tau - x)}} \tag{1}$$

and, as previously said, it gives the degree of determinism (or stochasticity) of the system. Note that the choice of this sigmoid function is empirical: several

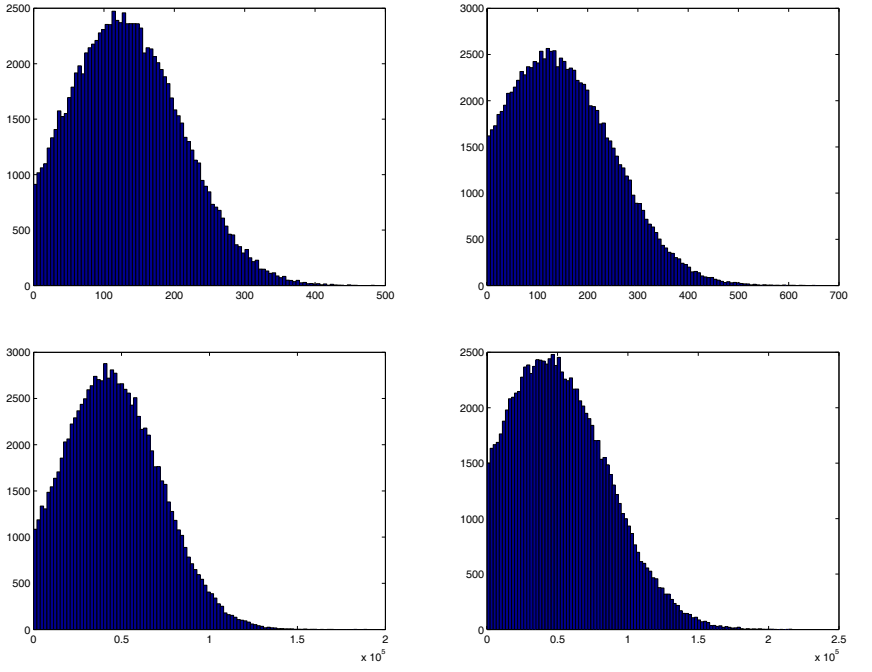


Fig. 1. Histograms of random sequences of 100000 samples with: mean 123 and variance $0.5 \cdot 123^2$ (upper left), mean 123, variance 123^2 (upper right), mean 42000, variance $0.5 \cdot 42000^2$ (lower left) and mean 42000, variance 42000^2 (lower right)

other functions can be employed (for example, the steepness of the sigmoid can be increased by using $b > e$, such as 8, instead of the Napier's base e of the exponential, or it can be decreased by using $b < e$, such as 2).

In Figure 2 two examples of the sigmoid function (1) are represented; on the left the threshold τ is set to 5000, whereas on the right it is set to 450. We can see that when the population size equals the threshold τ we have a strategy that is half deterministic and half stochastic, for this reason it may be better to consider the following sigmoid function:

$$d = \frac{1}{1 + 4^{\frac{20}{\tau'}(\tau' - x)}} \tag{2}$$

where $\tau' = 0.9\tau$, that is, the 90% of τ.

This newly defined threshold function is shown in Figure 3 (right). It is possible to appreciate that when the population size equals the threshold τ the strategy is deterministic at more than 95%. Although further investigation on the threshold function is needed, in the experiments described in the next section the modified sigmoid has been used.

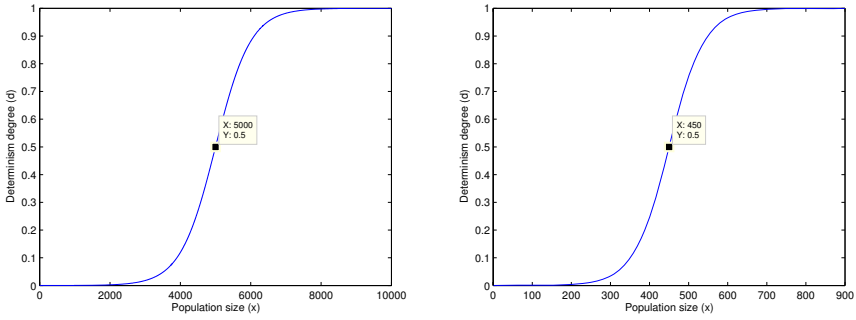


Fig. 2. Sigmoid function (1): population range 0–100000 $\tau = 5000$ (left); population range 0–900 $\tau = 450$ (right)

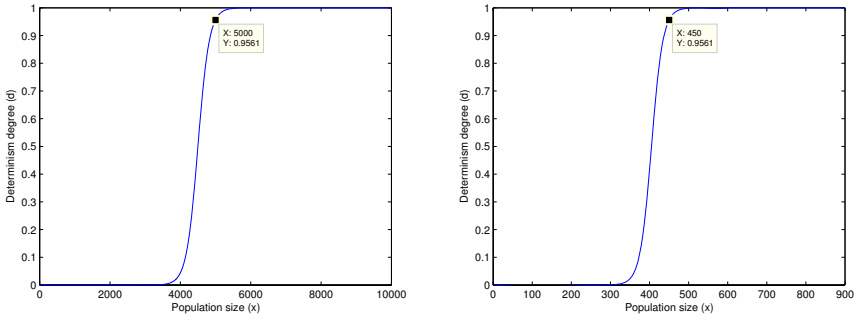


Fig. 3. Sigmoid function (2): population range 0–100000 $\tau' = 5000$ (left); population range 0–900 $\tau' = 450$ (right)

2.3 h-MPA

The idea of the algorithm is to apply each rule in a deterministic way whenever the population involved in it is bigger than a predetermined threshold; in turn, a rule is applied in a stochastic way whenever the population it deals with is below the threshold. For each rule, the total (i.e., not weighted by reaction maps) amount of the bottleneck is reckoned (the bottleneck of a reaction is the reactant whose total amount in the system is the lowest one when compared with the amounts of all other reactants of the rule) and a sigma function is calculated on it in order to obtain the determinism degree of the rule. Then, according to this determinism degree, the rule is applied partially in a deterministic way and partially in a stochastic way.

Let us assume a system specified by means of n rules $r_1, ..., r_n$, defined over the alphabet $\mathcal{A}$, initially in a state $\mathbf{s_0}$, and let τ be the deterministic degree threshold discussed previously. If we denote with $d_1(\mathbf{s}_i), ..., d_n(\mathbf{s}_i)$ the determinism degrees of each of the n rules (calculated as we will see in a while), we can express the dynamics of the system as the sequence $\mathbf{s}_0, \mathbf{s}_1, ...$ where a transition from a generic state $\mathbf{s}_i$ to the next one can be calculated by computing both the

stochastic and deterministic variation for all the rules and then weighting them according to the corresponding determinism degree. In particular, given a rule r_j with $1 \leq j \leq n$, its variation induced on the state $\mathbf{s}_i$, $\delta_{r_j}(\mathbf{s}_i)$, can be calculated as:

$$\delta_{r_j}(\mathbf{s}_i) = d_j(\mathbf{s}_i) \cdot MPA_j(\mathbf{s}_i) + (1 - d_j(\mathbf{s}_i)) \cdot s\text{-}MPA_j(\mathbf{s}_i)$$

where $MPA_j(\mathbf{s}_i)$ is the deterministic application of rule r_j to the state $\mathbf{s}_i$ while $s\text{-}MPA_j(\mathbf{s}_i)$ is the stochastic application of rule r_j to the state $\mathbf{s}_i$.

A transition from a state $\mathbf{s}_i$ to the next one $\mathbf{s}_{i+1}$ by means of the *hybrid* metabolic algorithm can be described by the following meta-code:

Step 0a: *Deterministic reaction maps computation.* The set of deterministic reaction maps is calculated in the current state:

$$F^D_{r_j}(\mathbf{s}_i) \quad \forall j = 1, ..., n\,.$$

Let us denote with $\mathcal{F}^D(\mathbf{s}_i)$ the set of all deterministic reaction maps in the state $\mathbf{s}_i$.

Step 0b: *Stochastic reaction maps computation.* The set of stochastic reaction maps is calculated in the current state:

$$F^S_{r_j}(\mathbf{s}_i) = RND(F^D_{r_j}(\mathbf{s}_i), 0.5 \cdot (F^D_{r_j}(\mathbf{s}_i))^2) \quad \forall j = 1, ..., n$$

where $RND(a, b)$ denotes a gaussian distributed random number, computed as seen before, with mean a and variance b, for $a, b \in \mathbb{R}$. Note that, as similar to what is made in the τ-leap method, while doing this calculation we have implicitly assumed constancy of the state [8].

Let us denote with $\mathcal{F}^S(\mathbf{s}_i)$ the set of all stochastic reaction maps in the state $\mathbf{s}_i$.

Step 1: *Single rule variations.* Deterministic and stochastic variations of each rule to each object of the system are computed.

For each couple (r_j, X) with $j = 1, ..., n$ and $X \in \mathcal{A}$, assuming each rule to have the form $r_j = \alpha_j \rightarrow \beta_j$:

o) If $X \notin \alpha_j$ AND $X \notin \beta_j$, then set both the deterministic and stochastic variation induced by r_j on X respectively to:

$$\delta^D_{r_j,X}(\mathbf{s}_i) = 0$$

$$\delta^S_{r_j,X}(\mathbf{s}_i) = 0$$

and goto the step o) of the next couple (if any), otherwise goto step i).

i) Calculate the rate d_j as:

1. Find the bottleneck[2] of reaction r_j:

$$X = \min_{Y \in \alpha_j} q(Y).$$

[2] Other choices for the bottleneck calculation are also reasonable; as it takes into account the size of the population involved in each rule instead of the global size of populations, the choice made here simplifies the algorithm.

2. The total amount of the bottleneck is calculated

$$x = q(X).$$

3. The deterministic rate is computed

$$d_j = \frac{1}{1 + 4^{\frac{20}{0.9\tau}(0.9\tau - x)}}.$$

ii) Calculate the deterministic variation induced by r_j on X, $\delta^D_{r_j,X}(\mathbf{s}_i)$ as in the standard metabolic algorithm with reaction maps taken from $\mathcal{F}^D(\mathbf{s}_i)$.

iii) Calculate the stochastic variation induced by r_j on X, $\delta^S_{r_j,X}(\mathbf{s}_i)$ as in the standard metabolic algorithm with reaction maps taken from $\mathcal{F}^S(\mathbf{s}_i)$.

Step 2: *Global variations and system update.* The global deterministic $\Delta^D_X(\mathbf{s}_i)$ and stochastic $\Delta^S_X(\mathbf{s}_i)$ variations are calculated by the weighted sum of all single rules contributions, $\forall X \in \mathcal{A}$:

$$\Delta^D_X(\mathbf{s}_i) = \sum_{j=1}^{n} \delta^D_{r_j,X}(\mathbf{s}_i) \cdot d_j$$

$$\Delta^S_X(\mathbf{s}_i) = \sum_{j=1}^{n} \delta^S_{r_j,X}(\mathbf{s}_i) \cdot (1 - d_j)$$

$$X_{i+1} := X_i + \Delta^D_X(\mathbf{s}_i) + \Delta^S_X(\mathbf{s}_i)$$

where X_i denotes the amount of species X in configuration $\mathbf{s}_i$.

Note that in the case of systems dealing with populations instead of concentrations a rounding policy has to be devised.

3 Case Studies

The case studies presented in this section have been implemented using Matlab, and resulted in numerical simulations whose computation typically took some seconds.

The first case study is the BZ reaction, that is discussed in two distinct variants, while the second case study is the Lotka-Volterra predator-prey system.

3.1 BZ (Model 1)

The first Brusselator model is composed by the following set of rewriting rules and deterministic reaction maps:

$$\begin{array}{lcll} r1: & \lambda & \longrightarrow & X & F_{r1} = 10 \\ r2: & XXY & \longrightarrow & XXX & F_{r2} = 900\, q(X)^2 q(Y) \\ r3: & X & \longrightarrow & Y & F_{r3} = 200\, q(X) \\ r4: & X & \longrightarrow & \lambda & F_{r4} = 5\, q(X). \end{array}$$

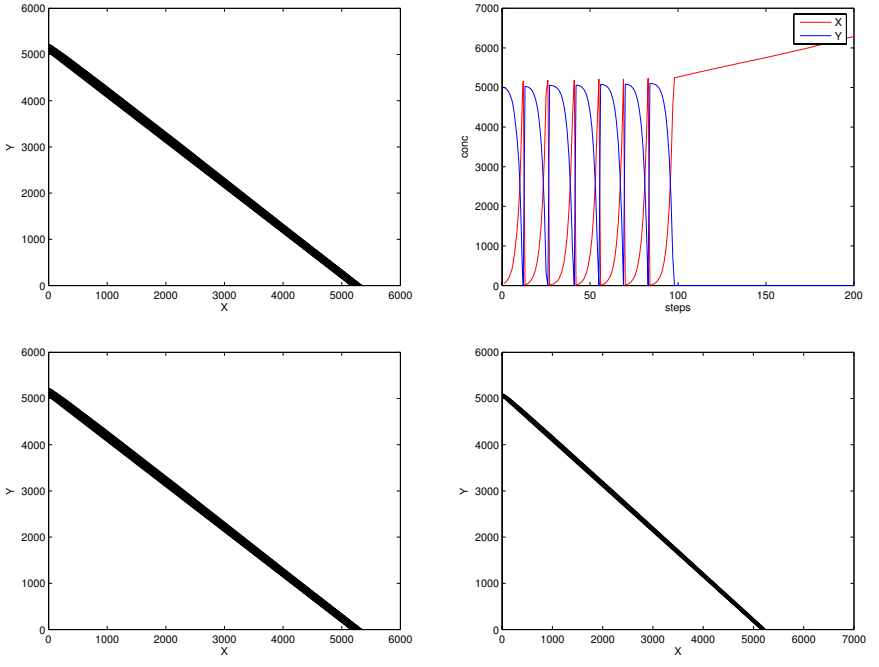


Fig. 4. Two realisations of a hybrid simulation of the Brusselator ($\tau = 2000$, deterministic rate of $r1$ set to 0.5). Corresponding phase space representations are depicted on the lower part of the figure.

The initial amount of objects X and Y are set respectively to 50 and 5000 and both procedures deal with integer objects (the contribution of all rules are floored to the nearest small integer). The deterministic rate that multiplies the variation obtained by the rule is a real value in general, hence for this reason we can have real values in the dynamics.

Note that, due to the fact that λ is not a population, we need a strategy to deal with rule $r1$, that is, we can decide to have a either a completely deterministic feeding of the system, a completely stochastic one, or every degree of determinism.

The algorithm can provide several behaviors depending on the value of the determinism threshold τ. It can show a completely deterministic dynamics if the threshold of determinism is set to 0, or conversely a completely stochastic one, in the case of the threshold $\tau \to \infty$ (or at least larger than the maximum population size in the whole simulation) [3,12]. Hybrid solutions are obtained for intermediate thresholds (see Figure 4, where $\tau = 2000$ and, on the right, we can observe that the dynamics shows a slight stochasticity in the first instants and then the oscillation is suppressed).

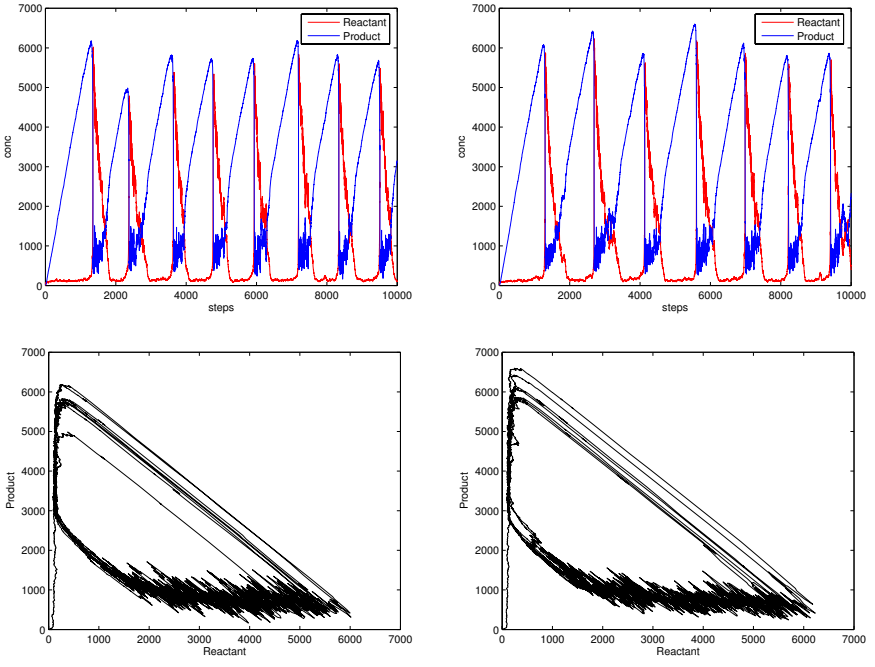


Fig. 5. Two realizations of completely stochastic simulations of the Brusselator ($\tau = 10^{99}$). Corresponding phase plots are shown in the lower figures.

3.2 BZ (Model 2)

The second Brusselator model deals with the following set of rewriting rules and deterministic reaction maps:

$$\begin{aligned}
r1 &: A \longrightarrow AX & F_{r1} &= 10^{-3} \\
r2 &: X \longrightarrow Y & F_{r2} &= 50 \\
r3 &: Y \longrightarrow X & F_{r3} &= 2.5 \cdot 10^{-5}\, q(X)^2 \\
r4 &: X \longrightarrow \lambda & F_{r4} &= 5 \\
r5 &: X \longrightarrow X & F_{r5} &= 1000 \\
r6 &: Y \longrightarrow Y & F_{r6} &= 1000 \\
r7 &: A \longrightarrow A & F_{r7} &= 1000.
\end{aligned}$$

The initial amounts for X and Y are respectively set to 1 and 10, while the constant feeding element A has an amount set to $5 \cdot 10^6$. The rate parameters have been taken from [7]. Moreover, rounding has not been used.

Two completely stochastic realizations are depicted in Figure 5, where the phase space is shown in the lower plots. Two hybrid realizations using different thresholds are depicted in Figure 6.

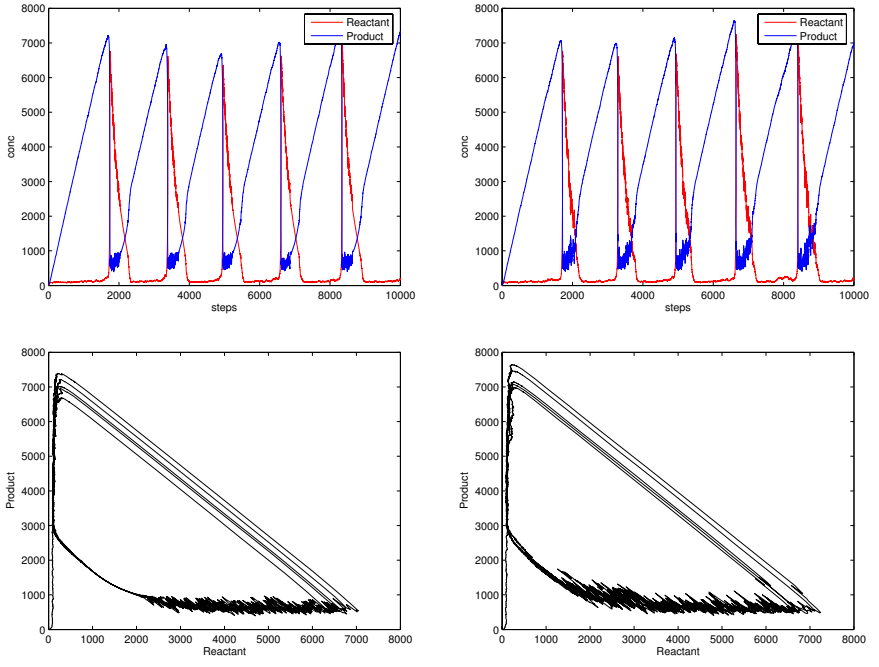


Fig. 6. Two hybrid simulation of the Brusselator ($\tau = 1000$ and $\tau = 3000$ respectively for the left hand part and right hand part of the figure). Corresponding phase plots are shown in the lower figures.

3.3 Lotka-Volterra

The Lotka-Volterra metabolic rewriting system is composed by the following set of rewriting rules and deterministic reaction maps [2]:

$$
\begin{array}{lll}
r1: & X \xrightarrow{k1} XX & F_{r1} = 3 \cdot 10^{-3} \\
r2: & XY \xrightarrow{k2} YY & F_{r2} = 4 \cdot 10^{-6} \cdot \max(X(k), Y(k)) \\
r3: & Y \xrightarrow{k3} \lambda & F_{r3} = 3 \cdot 10^{-3}\, q(X) \\
r4: & X \longrightarrow X & F_{r4} = 5 \\
r5: & Y \longrightarrow Y & F_{r5} = 5.
\end{array}
$$

Moreover, the initial populations of both predators (Y) and preys (X) are set to 900 and, for each species, an inertia (i.e., specified by rules of the type $X \longrightarrow X$ accounting for objects that cannot react in the considered instant) equal to 5 is also considered. Note that no rounding in the population dynamics is performed in this case. As usual, we can have completely deterministic dynamics [2] as well as a completely stochastic dynamics (see Figure 7). Hybrid behaviors can be obtained for intermediate values of τ (see Figure 8). An interesting case has arisen in a simulation using $\tau = 800$: in this simulation the randomness has led

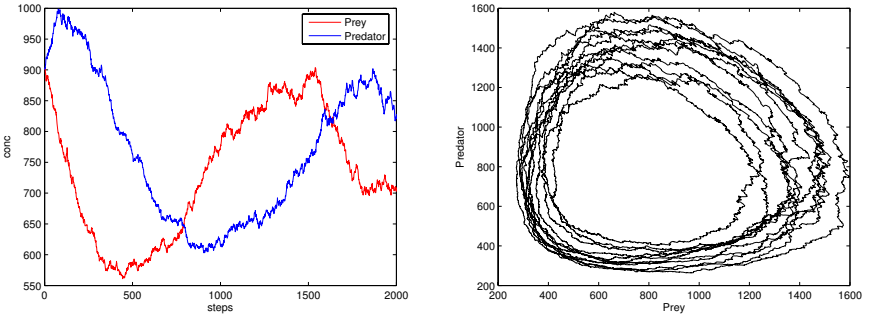


Fig. 7. Completely stochastic simulation of the LV system ($\tau = 10^{20}$), both evolution (left) and phase (right)

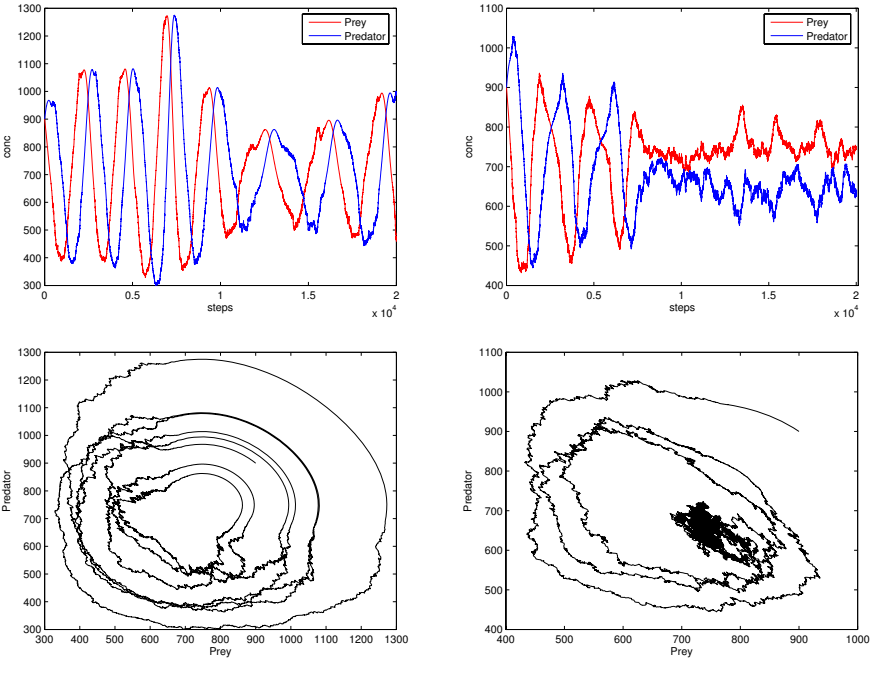


Fig. 8. Hybrid simulation of the LV system: $\tau = 700$ (left); $\tau = 800$ (right). Corresponding phase planes are shown in the lower figures.

the system to rapidly fall toward the steady-state. Of course, this behavior is not directly related to the choice of the threshold but, rather, to the random choices performed in the simulation. Otherwise, it would be impossible to obtain this dynamics from a purely deterministic simulation.

4 Conclusion

Although both deterministic and stochastic models for the simulation of biochemical systems have reached a good maturity, only a few things have been done in the direction of hybrid algorithms. We have shown here that this issue potentially leads to interesting dynamic representations, especially if coupled with the inherently versatile modeling formalism provided by MP systems.

Besides this, much still has to be done to make this strategy really competitive. Possible improvements in the short run may lead to a more suitable definition of the sigmoid function, and to a better tuned PRSG. In the medium and long run, alternative formalization of the h-MPA can be envisioned. In particular, a procedure accounting for two reaction maps for every rule, the former related to the deterministic, the latter to the stochastic behavior, may lead to rich realizations of a system evolution. For instance, it would be desirable to account for constants which depend on the regime, by using deterministic as well as stochastic rate constants that can be straightforwardly derived by exploiting the known relationships existing between the two [6].

References

1. L. Bianco, F. Fontana, G. Franco, and V. Manca. P systems for biological dynamics. In G. Ciobanu, G. Păun, and M.J. Pérez-Jiménez, editors, *Applications of Membrane Computing*, pages 81–126. Springer, 2006.
2. L. Bianco, F. Fontana, and V. Manca. Reaction-driven membrane systems. In L. Wang, K. Chen, and Y.-S. Ong, editors, *Advances in Natural Computation, First International Conference, ICNC 2005, Changsha, China, August 27-29, 2005, Proceedings, Part II*, volume 3611 of *Lecture Notes in Computer Science*, pages 1155–1158. Springer, 2005.
3. L. Bianco, F. Fontana, and V. Manca. P systems with reaction maps. *International Journal of Foundations of Computer Science*, 17(1):27–48, 2006.
4. Y. Cao, D. Gillespie, and L. Petzold. Avoiding negative populations in explicit Poisson tau-leaping. *J. of Chemical Physics*, 2005(123), 2005.
5. G. Ciobanu, G. Păun, and M.J. Pérez-Jiménez, editors. *Applications of Membrane Computing*. Springer, Berlin, 2006.
6. D.T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. of Computational Physics*, 22:403, 1976.
7. D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. of Physical Chemistry*, 81(25):2340–2361, 1977.
8. D.T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *J. of Chemical Physics*, 115(4):1716–1733, 2001.
9. E.L. Haseltine and J.B. Rawlings. Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics. *J. of Chemical Physics*, 117(15):1357–1372, 2002.
10. R. Illner, C.S. Bohun, S. McCollum, and T. van Roode. *Mathematical Modelling*. American Mathematical Society, Providence, RI, 2005.
11. V. Manca. Topics and problems in metabolic P systems. In G. Păun and M.J. Pérez-Jiménez, editors, *Proc. of the Fourth Brainstorming Week on Membrane Computing (BWMC4)*, Sevilla, Spain, February 2006. Fenix Editora.

12. V. Manca, L. Bianco, and F. Fontana. Evolutions and oscillations of P systems: Applications to biological phenomena. In G. Mauri, G. Păun, M.J. Pérez-Jiménez, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 5th International Workshop, WMC 2004*, volume 3365 of *Lecture Notes in Computer Science*, pages 63–84. Springer, 2005.
13. G. Păun. *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
14. G. Păun and G. Rozenberg. A guide to membrane computing. *Theoretical Computer Science*, 287:73–100, 2002.
15. M.J. Pérez-Jiménez and F.J. Romero-Campero. P systems: a new computational modelling tool for systems biology. *Transactions in Computational Systems Biology*, 2006. In press.
16. D. Pescini, D. Besozzi, G. Mauri, and C. Zandron. Dynamical probabilistic P systems. *International Journal of Foundations of Computer Science*, 17(1):183–194, 2006.
17. H. Salis and Y. Kaznessis. Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions. *J. of Chemical Physics*, 122:1–13, 2005.
18. E. Schrödinger. *What is life? With Mind and Matter and Autobiographical sketches*. Cambridge University Press, 1967.
19. T.E. Turner, S. Schnell, and K. Burrage. Stochastic approaches for modelling in vivo reactions. *Computational Biology and Chemistry*, 2004(28):165–178, 2004.
20. X. Zhang. A hybrid algorithm for determining protein structure. *IEEE Intelligent Systems*, 9(4):66–74, 1994.

Towards a P Systems Pseudomonas Quorum Sensing Model

Luca Bianco[1], Dario Pescini[2], Peter Siepmann[3], Natalio Krasnogor[3], Francisco J. Romero-Campero[4], and Marian Gheorghe[5]

[1] Department of Computer Science, University of Verona
Strada Le Grazie 15, 37134 Verona, Italy
bianco@sci.univr.it
[2] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
pescini@disco.unimib.it
[3] School of Computer Science and Information Technology
University of Nottingham
Jubilee Campus, Nottingham, NG81BB, UK
{Peter.Siepmann, Natalio.Krasnogor}@nottingham.ac.uk
[4] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville, Avda. Reina Mercedes, 41012 Sevilla, Spain
fran@cs.us.es
[5] Department of Computer Science, The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
M.Gheorghe@dcs.shef.ac.uk

Abstract. Pseudomonas aeruginosa is an opportunistic bacterium that exploits quorum sensing communication to synchronize individuals in a colony and this leads to an increase in the effectiveness of its virulence. In this paper we derived a mechanistic P systems model to describe the behavior of a single bacterium and we discuss a possible approach, based on an evolutionary algorithm, to tune its parameters that will allow a quantitative simulation of the system.

1 Introduction

The quorum sensing is a particular form of cell-to-cell communication in bacteria which exploits the concentration of a particular molecule, called signal, to "sense" the population density of the colony. The quorum sensing regulatory network is used by the individuals of the colony for collective synchronization and therefore for a coherent control over the gene expression. In Pseudomonas aeruginosa this mechanism is responsible for the effectiveness of the virulence of this bacterium [14,23,10,9]. In fact, a single bacterium starts to express his virulence factors only when it senses that the bacteria population has reached a certain threshold level such that the host response will be inadequate.

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 197–214, 2006.

The activation of a complex cellular response is what distinguishes the quorum sensing as a communication regulatory circuit from other density dependent responses such as the metabolization or detoxification of small molecules.

The simplest quorum sensing network known in Gram-Negative bacteria is also the first one ever discovered [25,17]. It has been found in the Vibrio fischeri bacterium, also known as Photobacterium fischeri and is nowadays considered as the paradigm of this cell communication process. In this network two proteins and one signalling molecule are involved. The *R protein* is a transcriptional regulator, while the *I protein* is the synthase for the signalling molecule, also referred to as the *autoinducer*. An important role is also played by the confinement of the bacterial colony. The fact that the *autoinducer* molecule is not dispersed in the environment allows its diffusion inside the individuals and therefore its concentration sensing.

At low cell densities the *I protein* synthesizes the *autoinducer* at a basal rate and the signal freely diffuses outside the bacterium. The concentration of the signal inside each bacterium is increased by the combined effect of the confinement and the increase of the population. At this point, the binding of the *R protein* with the *autoinducer* becomes more likely. The binding of the signal molecules activates the *R protein* transcriptional regulator. Since the *I gene* is the target of the *R protein*, the bacterium starts to produce more and more signal. The regulation network signal autoinduces its transcription. In this way the high concentration of the *autoinducer* coordinates the transcription of all the genes that are target of the *R protein*.

The quorum sensing in Pseudomonas aeruginosa is more complex, nevertheless intriguing, since this bacterium uses two different quorum sensing systems which interact with each other.

The aim of this work is to provide a P system model [15,16] of the bacterium Pseudomonas aeruginosa quorum sensing focusing on the communication mechanisms. The parameters of the model will be tuned using an evolutionary algorithm. Our long term aim is to reproduce the characteristic behavior of the quorum sensing in Pseudomonas aeruginosa, namely, the switch between two distinct stable steady solutions: the first describing the behavior of the non-quorated bacterium (i.e., with low levels of autoinducer), the second modeling its quorated behavior (i.e., the behavior obtained with high concentration of the autoinducer molecule). Once the model will be entirely defined several simulations with different strategies [6,20,18] will be run.

First of all, we address the modeling of the internal dynamics of one single bacterium, tuning its kinetic constants in a way ensuring its non-quorated behavior. At a later stage, we intend to exploit compartmentalization of P systems to model a colony of bacteria each of them internally specified according to the same set of kinetic constants. In this respect we will extend the current model to a Population P systems approach [3] that has been already used to express some aspects of quorum sensing in bacterium Pseudomonas aeruginosa [22] and for self-assembly problems [4].

2 An Initial Model

The first stage of our investigation is intended to describe the quorum sensing related network of each bacterium to capture its main features into a mechanistic model. The quorum sensing internal pathway of each bacterium is taken from models discussed in [11,9] and a graphical representation of all elements involved in it, as well as some relevant relationships between them, are depicted in Figure 1.

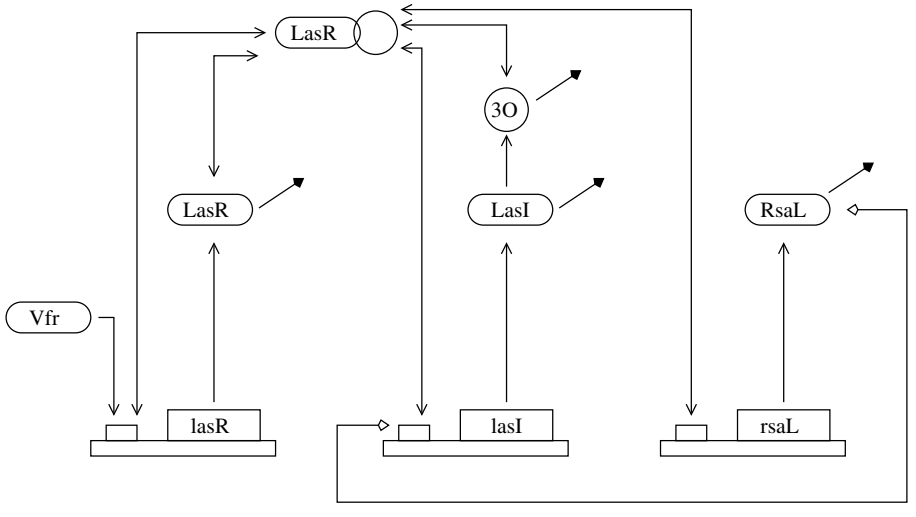


Fig. 1. The Pseudomonas quorum sensing model analyzed here (from [9]). Note that double arrows denote reversible reactions, bold ones the degradation process and the empty ones the inhibitory process.

According to this model, the quorum sensing pathway comprises two interconnected signalling cascades. The main elements involved in the first one are proteins $LasR$, $RsaL$, $LasI$ (as well as the genes involved in their production), the autoinducer molecule 3-oxo-$C12$-HSL and the active complex $LasR$-3-oxo-$C12$-HSL. The key elements of the second system are the proteins $RhlR$ and $RhlI$ (as well as the genes involved in their production), the autoinducer molecule $C4$-HSL and the active complex $RhlR$-$C4$-HSL. The first one of the two signalling cascades is called *las* system because it was shown to regulate the expression of LasB elastase. This pathway regulates other virulence factors such as $LasA$ protease, exotoxin A, alkaline protease A as well as the expression of at least two genes of the *xcp* secretory pathway. The *las* pathway is positively controlled by $GacA$ and Vfr whereas it is inhibited by $RsaL$ that, in turn, is positively regulated by the active complex $LasR$-3-oxo-$C12$-HSL and whose role is to repress the transcription of the $lasI$ gene.

The second signalling system involved in the model is named *rhl* system because it controls the expression of rhamnolipid via the production of $rhlAB$

operon. The autoinducer molecule in this case is $C4\text{-}HSL$ and the active complex is $RhlR\text{-}C4\text{-}HSL$. It has been shown that this cascade is necessary for the production of some virulence factors like $LasB$ elastase and $LasA$ protease, as well as pyocyanin, cyanide and alkaline protease. For this reason this signalling system is also known as *vsm* (virulence secondary metabolites).

Although the corresponding autoinducing molecules are highly selective (and thus not interchangeable at all), several interconnections between the *las* and the *rhl* pathways of the quorum sensing in Pseudomonas aeruginosa are known. One link between them has been already mentioned and it is constituted by the $LasB$ elastase, that needs both $LasR\text{-}3\text{-}oxo\text{-}C12\text{-}HSL$ and $RhlR\text{-}C4\text{-}HSL$ for its production. More interestingly, the *las* system is at a higher level in the hierarchical regulatory cascade, in fact $LasR\text{-}3\text{-}oxo\text{-}C12\text{-}HSL$ can activate the expression of the $rhlR$ gene. In addition, the active complex $LasR\text{-}3\text{-}oxo\text{-}C12\text{-}HSL$ can bind to $RhlR$ preventing it to form the complex $RhlR\text{-}C4\text{-}HSL$.

2.1 The Differential Equation Model

Many models for the quorum sensing in the Pseudomonas aeruginosa are presented in literature and usually they approach the phenomenon from two different angles. The first one describes the colony behavior by summarizing individual dynamics as a state change avoiding a precisely detailed representation of each of the bacterium quorum sensing networks [24,1]. The second one describes in a more detailed fashion the quorum sensing pathway for each bacterium with the purpose to model the emergent behavior of the whole colony [11].

We think that the P system framework is particularly suitable for this second approach. In fact, the modularity, the compartmentalization, the hierarchical structure and the rewriting rules (all features of P systems [16]) allow a convenient description of this reality.

In [11] a model of the *las* signalling system has been devised, but no description is given of the rhl system. The graphical description of the quorum sensing pathway depicted in Figure 1 has been translated into the set of eight differential equations presented in the next page. The correspondence between differential equation symbols and elements in the pathway are summarized in Table 1.

The production of the activated complex P by means of the autoinducer and the $LasR$ protein (whose expression is given by the product of the constitutive elements concentrations with a rate k_{RA}: $k_{RA}RA$) is an example of how cooperative contributions are obtained in the differential equations approach by means of the mass action law.

Basal rates productions and degradations are also taken into account, an example of the former being the $k_1 r$ element giving the basal production of $LasR$ protein (R), while an example of the latter is the degradation of the active complex (P) represented by the element $k_P P$. The production of messenger RNAs from the corresponding genes is modeled with a Michaelis-Menten-like dynamics depending on the concentration of the promoting factor, as it happens in the case of the production of $lasR$ and $rsaL$ mRNAs (respectively r and s), the

first modeled by $V_r \frac{P}{K_r + P}$ and the second by $V_s \frac{P}{K_s + P}$. The production of $lasI$ mRNA (l) is also down-regulated by the presence of $RsaL$ protein (S) and this is modeled by $V_l \frac{P}{K_l + P} \frac{1}{K_S + S}$, in which the Michaelis-Menten-like dynamics is attenuated by an inversely proportional function of the $RsaL$ concentration.

$$\begin{aligned}
\frac{dP}{dt} &= k_{RA} RA - k_P P \\
\frac{dR}{dt} &= -k_{RA} RA + k_P P - k_R R + k_1 r \\
\frac{dA}{dt} &= -k_{RA} RA + k_P P + k_2 L - k_A A \\
\frac{dL}{dt} &= k_3 l - k_L L \\
\frac{dS}{dt} &= k_4 s - k_S S \\
\frac{ds}{dt} &= V_s \frac{P}{K_s + P} - k_s s \\
\frac{dr}{dt} &= V_r \frac{P}{K_r + P} - k_r r + r_0 \\
\frac{dl}{dt} &= V_l \frac{P}{K_l + P} \frac{1}{K_S + S} - k_l l + l_0
\end{aligned} \tag{1}$$

Unfortunately, no value is known for the 21 kinetic constants present in the set of differential equations (1). To overcome this problem, in [11] several simplifying assumptions are considered, that lead to fewer equations and fewer parameters as well.

In the following we will describe a possible parameter estimation strategy to tackle this problem (see Section 4). The idea is to relay to this differential equations system as a "synthetic bio-experiment" used to confront our model to.

2.2 A First P Systems Model

Several attempts to simulate the quorum sensing in bacteria are present in P systems literature [5,19], but, as far as we know, none of them deals with the Pseudomonas aeruginosa bacterium.

Here we describe a direct P systems translation of the differential equation model previously discussed [11]. Formally, the Pseudomonas P system is

$$\Pi = (A, \mu, w, R)$$

Table 1. Variable-concentration correspondence between the differential formulation and the graphical description of the quorum sensing model of Pseudomonas aeruginosa (from [11])

Variable	Concentration
R	$LasR$
A	3-oxo-$C12$-HSL
P	$LasR$-3-oxo-$C12$-HSL
L	$LasI$
S	$RsaL$
r	$lasR\ mRNA$
l	$lasI\ mRNA$
s	$rsaL\ mRNA$

where:

- $A = \{geneR, geneL, R, A, P, L, S, r, l, s\}$ is the alphabet;
- $\mu = [\ ]_0$ is the membrane structure: since we address the single bacterium case, it contains the cellular membrane only;
- $w = geneR\, geneL$ is the initial configuration that comprises only $LasR$ and $LasI$ genes, thus is represented as the string;
- $R = \{r_1, \cdots, r_{18}\}$ is the set of the rules:

$$
\begin{aligned}
r_1 &: geneR \longrightarrow geneR + r \\
r_2 &: r \longrightarrow \lambda \\
r_3 &: r \longrightarrow r + R \\
r_4 &: P \longrightarrow P + r \\
r_5 &: R + A \longrightarrow P \\
r_6 &: P \longrightarrow R + A \\
r_7 &: P \longrightarrow P + s \\
r_8 &: s \longrightarrow \lambda \\
r_9 &: S \longrightarrow \lambda \\
r_{10} &: s \longrightarrow s + S \\
r_{11} &: P \longrightarrow P + l \\
r_{12} &: l \longrightarrow l + L \\
r_{13} &: l \longrightarrow \lambda \\
r_{14} &: geneL \longrightarrow geneL + l \\
r_{15} &: L \longrightarrow \lambda \\
r_{16} &: L \longrightarrow L + A \\
r_{17} &: A \longrightarrow \lambda \\
r_{18} &: R \longrightarrow \lambda
\end{aligned}
$$

Note that, symbols in A correspond to the variables of the differential equation and their correspondence to the biological reality is given in Table 1. Two new elements (i.e., $geneR$ and $geneL$) are introduced, which account for the genes involved in the basal production of the $LasR$ and $LasI$ mRNAs.

Each one of the rules in R is directly obtained from the differential description of the considered quorum sensing model. For examples, we can see that rule r_1 models the basal production of the $LasR$ mRNA, while rule r_2 expresses its degradation, moreover rules r_5 and r_6 describe the reversible reaction of the complex P formation by starting from its fundamental constituents R and A.

Due to the different level of abstraction in the representation of different parts of the model (as in the case of the Michaelis-Menten-like kinetics that are modelled with a higher level of abstraction than other components of the system), we cannot directly apply mechanistic algorithms [2] to this model. For this reason, we will apply to this set of rules only the strategy known as *Metabolic Algorithm* (for details refer to [6]), whose simulation results, together with some numerical solutions of the set of differential equations (1), are shown in Section 2.3 for different choices of parameters.

The metabolic algorithm simulation needs to specify a set of reaction maps, each one associated in a one-to-one manner to the rules of R. Reaction maps [6] are functions defined over the state of the system (i.e., multiplicity or concentration of all elements of the system depending on the case), that are used by the Metabolic algorithm to allocate objects to rules. For example, as we will see in a while, F_{r_1}, that is the reaction map of rule r_1, is simply the constant rate of production of LasR mRNA. We can have more complicated reaction maps, as in the case of rule r_4 that takes into account the Michaelis-Menten-like production of the LasR mRNA elicited by the $LasR$-3oxo-C12-HSL complex. As in the case of the rules, that specify the physical interactions and connections between the elements of the modeled reality, we can obtain this information from the differential equation formulation. The set of reaction maps employed in our simulations are the following:

$$
\begin{array}{llll}
F_{r_1} = r_0 & & F_{r_2} = k_r & \\
F_{r_3} = k_1 & & F_{r_4} = \frac{V_r}{K_r+P} & \\
F_{r_5} = k_{RA} & & F_{r_6} = k_P & \\
F_{r_7} = \frac{V_s}{K_s+P} & & F_{r_8} = k_s & \\
F_{r_9} = k_4 & & F_{r_{10}} = k_S & \\
F_{r_{11}} = \frac{V_l}{(K_l+P)\cdot(K_S+S)} & & F_{r_{12}} = k_3 & \\
F_{r_{13}} = k_l & & F_{r_{14}} = l_0 & \\
F_{r_{15}} = k_L & & F_{r_{16}} = k_2 & \\
F_{r_{17}} = k_A & & F_{r_{18}} = k_R &
\end{array}
\tag{2}
$$

Note that all reaction maps are constant apart from three of them. We have already discussed the meaning of the reaction map associated to rule r_4; analogous considerations hold for F_{r_7} as well. More interesting is the reaction map associated to rule r_{11} that takes into account the inhibitory effect of $RsaL$ protein on the production of the $lasI$ mRNA.

Remarkably, the method allows the current description of different parts of the system at different abstraction levels; moreover it is still applicable if all reaction maps are constant, a condition required by mechanistic algorithms.

In the following some simulation results are shown, as well as the numerical solution of the differential equation system, for some chosen parameters.

2.3 Simulation Results

Here we show how the same model-reality can be described with two different approaches. As mentioned before, we do not have precise values for the model parameters. For this reason, as a first comparison attempt, we make a completely fictitious choice for them. As a further work, we plan to adopt some automatic way for the parameter estimation (see Section 4 for more details). The initial choice of parameters is here shown, and all the subsequent changes to this initial parameter set will be explicitly mentioned:

$$\begin{array}{llll} k_{RA} = 10 & & k_P = 2 \\ k_R = 5 & & k_1 = 1 \\ k_2 = 1 & & k_A = 1 \\ k_3 = 1 & & k_L = 1 \\ k_4 = 1 & & k_S = 1 \\ V_s = 1 & & K_s = 1 \\ k_s = 0.5 & & V_r = 1 \\ K_r = 1 & & k_r = 1 \\ r_0 = 1 & & V_l = 1 \\ K_l = 1 & & k_l = 1 \\ l_0 = 1 & & K_S = 1 \end{array} \tag{3}$$

The lack of biological information makes this choice completely arbitrary and prevents us to compute the dynamics of the system by means of stochastic algorithms such as the Gillespie one [12,13], Dynamical Probabilistic P Systems [20] or the Multi-compartmental Gillespie [18].

In this section we compare the dynamics generated by the metabolic algorithm with the solutions obtained for the corresponding differential equation system. Figure 2 depicts the case in which parameters are chosen according to (3). The dynamics of each species reaches a steady state in both approaches, but the relative position of the species is different and this leads to two distinct system dynamics. Moreover, the time of the two systems differs; in the solution of the differential equation system this is measured in arbitrary units (due to the arbitrary choice of parameters), while in the model based on P systems the time is measured in steps of system evolution. In Figure 3 the choice of $V_l = 0$ switches off rule r_{11} of the P system model and in this case the results of the two different approaches qualitatively match each other. Finally, the last choice of parameters is aimed at obtaining a quorum sensing consistent behavior, that is, in the case of a single bacterium in the environment it should not quorate and thus the concentration of the complex P should reach the basal rate. Accordingly, we set K_{RA} to the value 0.1. In this case, depicted in Figure 4, the dynamics produced by the two approaches is qualitatively similar again.

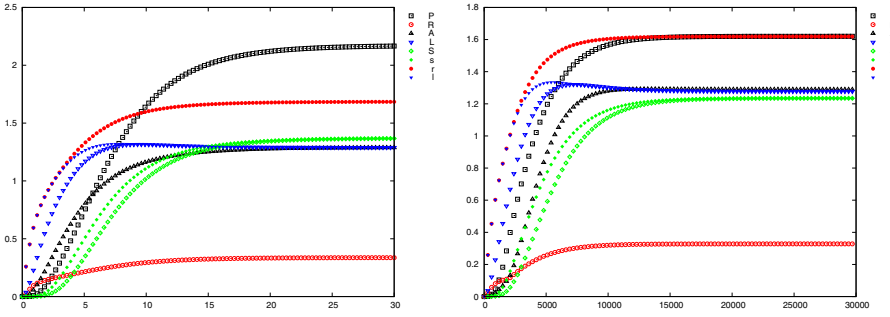


Fig. 2. Results for the quorum sensing model with parameters showed in (3) using ODE approach (left) and metabolic algorithm (right)

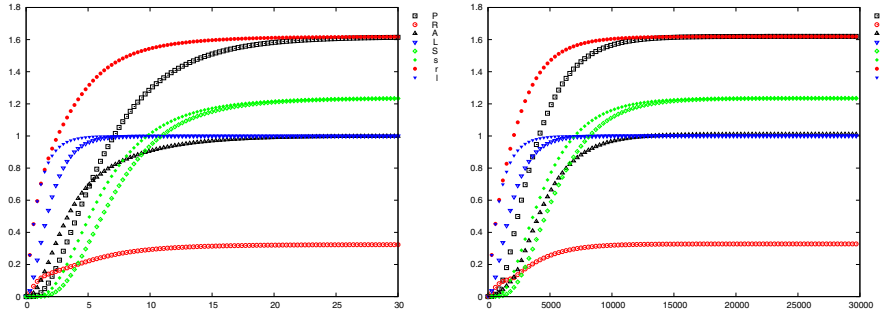


Fig. 3. Results for the quorum sensing model with $V_l = 0$ using ODE approach (left) and metabolic algorithm (right)

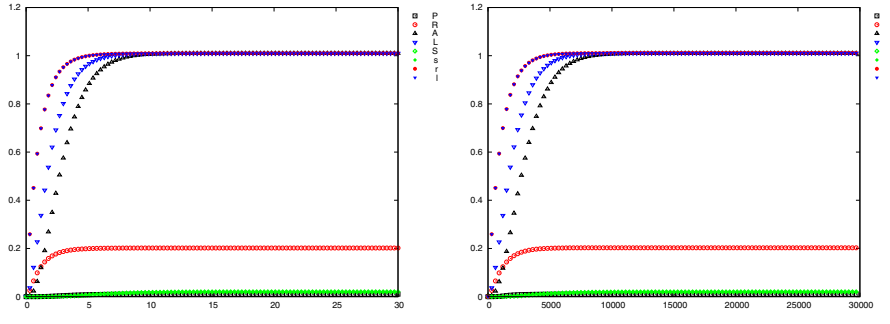


Fig. 4. Results for the quorum sensing model with parameters $k_{RA} = .1$ using ODE approach (left) and metabolic algorithm (right)

3 Towards a Detailed P Systems Model

Although the preliminary P system model described in Subsection 2.2 showed that we can obtain comparable results with the current models presented so

far, our intention is to refine the model defined above in order to allow the simulation of its dynamics by means of mechanistic approaches like Gillespie approach [12,13], Dynamical Probabilistic P Systems algorithm [20] or the Multicompartmental Gillespie method [18]. In addition, this model is completely driven by the set of differential equations and in same cases it is not entirely biologically accurate. For example, in the case of $rsaL$ mRNA production, when different from other mRNAs productions, it does not show any basal rate production. Moreover, it does not consider the binding of the transcription factor to the appropriate gene site necessary to start the transcription process of the DNA into the mRNA.

The formal description of the detailed P system model of Pseudomonas quorum sensing is the following:

$$\Pi = (A, \mu, w, R)$$

where:

- $A = \{Vfr, lasR_{gene}, Vfr.lasR_{gene}, mlasR, LasR, 3OHSL, LasR.3OHSL, LasR.30HSL.lasR_{gene}, lasI_{gene}, LasR.30HSL.lasI_{gene}, mlasI, LasI, rsaL_{gene}, LasR.3OHSL.rsaL_{gene}, mrsaL, RsaL, RsaL.lasI_{gene}\}$ is the alphabet;
- $\mu = [\,]_0$ is the membrane structure: since we address the single bacterium case, it contains the cellular membrane only;
- $w = Vfr^n \quad lasR_{gene} \quad lasI_{gene} \quad rsaL_{gene}$ is the initial configuration that comprises only the three genes and the protein Vfr that is needed to initiate the transcription and should be initialized at an high amount $n \in \mathbf{N}$;
- $R = \{r_1, \cdots, r_{28}\}$ is the set of the rules:

$$\begin{aligned}
r_1 &: Vfr + lasR_{gene} \xrightarrow{k_1} Vfr.lasR_{gene} + Vfr \\
r_2 &: Vfr.lasR_{gene} \xrightarrow{k_2} lasR_{gene} \\
r_3 &: Vfr.lasR_{gene} \xrightarrow{k_3} Vfr.lasR_{gene} + mlasR \\
r_4 &: mlasR \xrightarrow{k_4} \lambda \\
r_5 &: mlasR \xrightarrow{k_5} LasR + mlasR \\
r_6 &: LasR \xrightarrow{k_6} \lambda \\
r_7 &: LasR + 3OHSL \xrightarrow{k_7} LasR.3OHSL \\
r_8 &: LasR.3OHSL \xrightarrow{k_8} LasR + 3OHSL \\
r_9 &: 3OHSL \xrightarrow{k_9} \lambda \\
r_{10} &: LasR.3OHSL + lasR_{gene} \xrightarrow{k_{10}} LasR.3OHSL.lasR_{gene} \\
r_{11} &: LasR.3OHSL.lasR_{gene} \xrightarrow{k_{11}} LasR.3OHSL + lasR_{gene} \\
r_{12} &: LasR.3OHSL.lasR_{gene} \xrightarrow{k_{12}} LasR.3OHSL.lasR_{gene} + mlasR \\
r_{13} &: LasR.3OHSL + lasI_{gene} \xrightarrow{k_{13}} LasR.3OHSL.lasI_{gene} \\
r_{14} &: LasR.3OHSL.lasI_{gene} \xrightarrow{k_{14}} LasR.3OHSL + lasI_{gene}
\end{aligned}$$

$$
\begin{aligned}
r_{15} &: LasR.3OHSL.lasI_{gene} && \xrightarrow{k_{15}} LasR.3OHSL.lasI_{gene} + mlasI \\
r_{16} &: mlasI && \xrightarrow{k_{16}} \lambda \\
r_{17} &: mlasI && \xrightarrow{k_{17}} LasI + mlsaI \\
r_{18} &: LasI && \xrightarrow{k_{18}} \lambda \\
r_{19} &: LasI && \xrightarrow{k_{19}} LasI + 3OHSL \\
r_{20} &: LasR.3OHSL + rsaL_{gene} && \xrightarrow{k_{20}} LasR.3OHSL.rsaL_{gene} \\
r_{21} &: LasR.3OHSL.rsaL_{gene} && \xrightarrow{k_{21}} LasR.3OHSL + rsaL_{gene} \\
r_{22} &: LasR.3OHSL.rsaL_{gene} && \xrightarrow{k_{22}} LasR.3OHSL.rsaL_{gene} + mrsaL \\
r_{23} &: mrsaL && \xrightarrow{k_{23}} \lambda \\
r_{24} &: mrsaL && \xrightarrow{k_{24}} RsaL + mrsaL \\
r_{25} &: RsaL && \xrightarrow{k_{25}} \lambda \\
r_{26} &: RsaL + lasI_{gene} && \xrightarrow{k_{26}} RsaL.lasI_{gene} \\
r_{27} &: RsaL.lasI_{gene} && \xrightarrow{k_{27}} RsaL + lasI_{gene} \\
r_{28} &: RsaL.lasI_{gene} && \xrightarrow{k_{28}} RsaL.lasI_{gene} + mlasI
\end{aligned}
$$

where k_i, for $i = 1, \cdots, 28$, is the rate constant associated to the ith rule.

This system is depicted in Figure 5 where numbers next to arrows refer to the corresponding rules. Note that arrows with two numbers denote reversible reactions modeled in the P system description with two distinct rules.

To give some ideas on how the model has been built we explain in details the process that, starting from the $lasI_{gene}$, leads to the formation of the complex $LasR.3OHSL$, the remaining part of the model follows a similar derivation. The

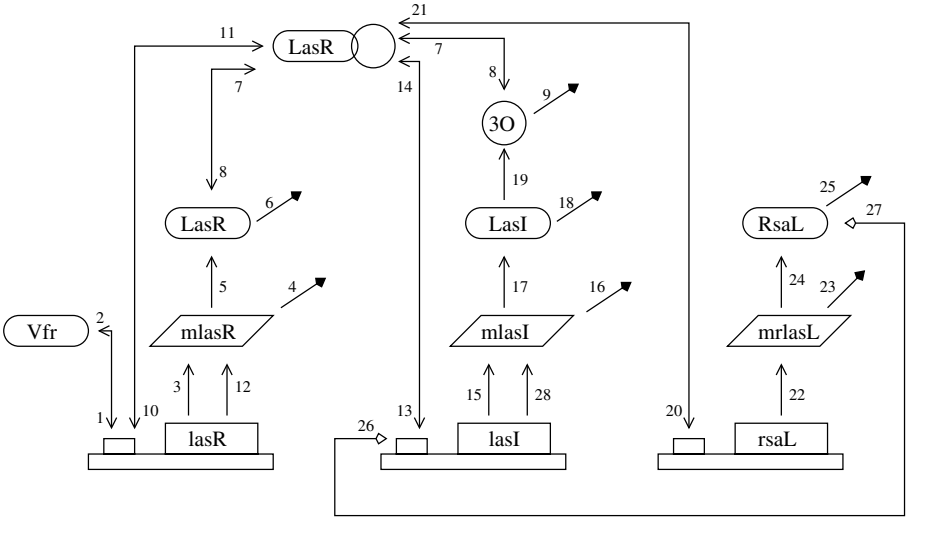


Fig. 5. The Pseudomonas quorum sensing detailed model analyzed here. The number next to each arrow refers to the corresponding P system rule.

production of $LasI$ mRNA ($mlsaI$) can be done in two ways depending on the transcription factor bound to the $lasI_{gene}$ gene. In fact, when $LasR.3OHSL$ binds to the $lasI_{gene}$ (rule r_{13}), it activates the transcription of $lasI_{gene}$ gene into $mlasI$ mRNA (rule r_{15}) with a rate k_{15}. The $RsaL$ protein can bind to the $lasI_{gene}$ gene as well (rule r_{26}), but in this case, the same transcription (modeled by rule r_{28}) has different rate k_{28}. Since the biology of the process tells us that $RsaL$ protein inhibits the mRNA production, we add the constraint that $k_{28} \ll k_{15}$. The $mlasI$ mRNA can either be degraded (rule r_{16}) either be translated into the $LasI$ protein (rule r_{17}). The latter can in turn be degraded (rule r_{18}) or it can produce the autoinducer molecule $3OHSL$ (rule r_{19}), that can bind to the $LasR$ protein and form the complex $LasR.3OHSL$ (rule r_7) or be degraded (rule r_9).

As far as we know, no value for the kinetic constant necessary for the simulation of this dynamics is known in literature. For this reason we plan to adopt some automatic tools for exploring the huge parameter space. In the following section we describe a genetic algorithm (GA) fitting approach.

4 Parameter Estimation

In previous sections we showed two alternative models of quorum sensing and *qualitatively* compared them against a differential equations based model. In this section we show, as a proof of concept, how P system models can be *quantitatively* fitted to observed data. In this proof of concept section we consider the ODE model as the golden standard against which the P system must be fitted. That is, the ODE is a proxy for a biological experiment from which we could measure a variety of molecular concentrations. In order to fit the P system models to the ODE's observed data we perform parameter optimization using an evolutionary algorithm (EA). Our EA has been specially developed for optimizing a range of design and manufacturing processes. It has been successfully tested on a variety of complex systems and nano-particles self-organization system [21]. Our evolutionary system is web-server based and can be tailored to solve a broad range of problems. The number and data types of genes in the chromosome, along with the parameters for the GA, including the users choice of selection, replacement, mutation and crossover mechanisms can be specified in the web-based configuration module. The later builds an XML script as output. This script, along with a plug-in style problem specification class, which most importantly includes the fitness function, configures the evolutionary algorithm to the specific problem at hand. The execution of the evolutionary algorithm can then be started and observed over the internet through a Java servlet. This evolutionary engine also caters for cpu-intensive optimization problems, like the one we investigate here, by distributing the execution of the algorithm on a large computer cluster. Moreover, the web-server also allows simultaneous executions of the evolutionary engines on different problems. The web-server can be accessed (under request) from `www.chellnet.org`. For a schematic representation of the evolutionary engine please see Figure 6.

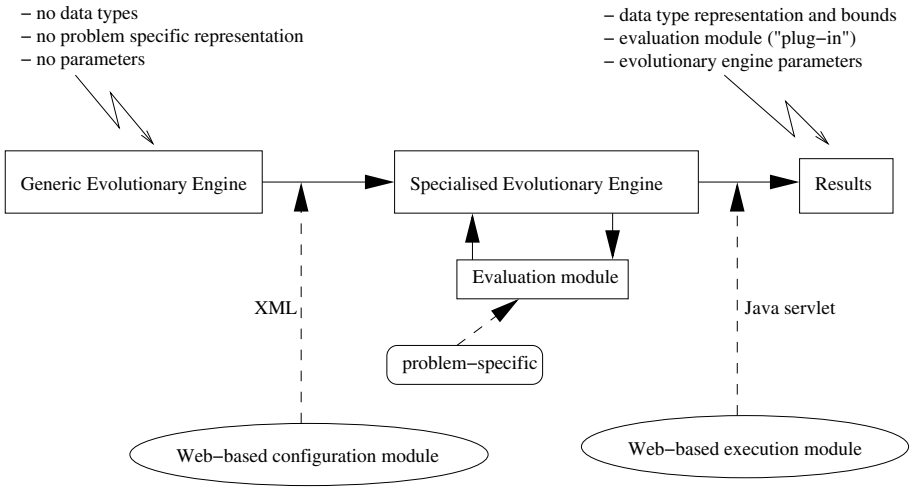


Fig. 6. The ChellNet Evolvable Chellware Engine

In what follows we describe the fitness function used to fit our P systems to the observed time series.

4.1 The Fitness Function

The evolutionary engine is used to adjust the parameters of the P system as to fit the observed target $w \in \mathbf{N}$ time series $\mathcal{S}_{tgt} = \{s^i_{tgt}\}_{i=1,\cdots,w}$ simultaneously, where each of the w time series corresponds to one of the species concentrations. In turn, the P system model generates w time series $\mathcal{S}= \{s^i\}_{i=1,\cdots,w}$. The evolutionary algorithm goal is to minimize the error between $\mathcal{S}_{tgt}$ and $\mathcal{S}$. Although simply put, this error must be done carefully as the sampling of the P system's $\mathcal{S}$ and that of $\mathcal{S}_{tgt}$ are different. If $s^i_{tgt} \in \mathcal{S}_{tgt}$, with (dropping the super-index for simplicity) $s_{tgt} = \{y(0), y(\varepsilon), y(2\,\varepsilon), \ldots, y(n\,\varepsilon)\}$ and ε the time step precision for $\mathcal{S}_{tgt}$, and $s = \{y'(0), \ldots, y'(t'_j), \ldots, y'(t'_m)\}$ there is no direct mapping from t'_j (in s) to $k\varepsilon$ (in s_{tgt}) for some $k \geq 1$ as the time interval simulated is not uniformly sampled under a Gillespie dynamics. In order to compute the error between a given $y'(t'_j)$ and a candidate $\hat{y}$ interpolated from s_{tgt} we need to interpolate the value $\hat{y}(t'_j)$ that s_{tgt} would take at t'_j. Note that the only point in time that is guaranteed to match in both time series is t_0, so we can obtain the index

$$k = \lfloor \frac{t'_j - t_0}{\varepsilon} \rfloor.$$

With the index k we can interpolate s_{tgt} between the time steps t_k and t_{k+1}:

$$q = \frac{y(t_{k+1}) - y(t_k)}{\varepsilon},$$

that is, the slope of the segment of line that runs between points $(t_k, y(t_k))$ and $(t_{k+1}, y(t_{k+1}))$. With q we can interpolate the value of s_{tgt} at time t'_j with

$$\hat{y}(t'_j) = y(t_k) + q\ (t'_j - t_k).$$

With this provision in mind the parameter learning problem becomes

$$min \sum_{s \in S_{tgt}} \frac{\sum_{\forall t'_j \in s^i, s^i \in S} \frac{|\hat{y}(t'_j) - y'(t'_j)|}{max\{\hat{y}(t'_j), y'(t'_j)\}}}{||s^i||}. \tag{4}$$

Eq. 4 is used by the evolutionary algorithm to fit the P system to the data. This fitness measure takes into account all the time series to be approximated and the quality of the sample of each time series.

4.2 A Case Study: The Michaelis-Menten Dynamics

In order to demonstrate the feasibility of automatically tuning a P system with an evolutionary algorithm we choose a simple case study: we apply the evolutionary algorithm to the problem of matching the kinetic constants of a Michaelis-Menten dynamics (MM). The MM dynamics is numerically obtained through a set of differential equations that simulate the following enzymatic reactions:

$$E + S \underset{k_1}{\overset{k_2}{\rightleftarrows}} ES \xrightarrow{k_3} E + P \tag{5}$$

where E represent the enzyme catalyzing the reaction transforming the substrate S into the product P. The reaction takes place in two different stages, the former being the reversible formation of the active complex ES, the latter being the production of P. All the details regarding the MM dynamics can be found in [7,8].

As mentioned above these reactions are modeled by means of the following set of differential equations:

$$\begin{aligned} \frac{d[S]}{dt} &= -k_1 E_0[S] + (k_1[S] + k_2)[ES] \\ \frac{d[ES]}{dt} &= k_1 E_0[S] - (k_1[S] + k_2)[ES] \\ \frac{d[P]}{dt} &= k_2[ES] \end{aligned} \tag{6}$$

where E_0 represents the concentration of the total amount of enzyme (i.e., the free enzyme plus that bounded to the substrate to form the complex ES), while, as usual in biochemistry, $[X]$ represent the concentration of the species X. The reactions (5) can be straightforwardly translated into a P system having only one compartment and three rules (each one referring to exactly one of the biochemical reactions mentioned), whose dynamics can be calculated by means of the Gillespie algorithm.

Fig. 7. Fitness progress of the parameter learning process. Best individual and average error in the population is shown.

Without loss of generality, we arbitrarily fix the three kinetic constants to $k_1 = 1000, k_2 = 1$ and $k_3 = 0.05$ and we numerically solve the differential equations. The initial conditions used are 0.001 M for the initial substrate S and $0.5 \cdot 10^{-3}$ M for the initial concentration of the enzyme E (no product P neither active complex ES is present at the beginning). We thus obtain three time series that represent the *target behavior* the P system must imitate. The evolutionary algorithm thus must coerce the P system to mimic as close as possible the MM dynamics (with an imaginary volume fixed to $1.67 \cdot 10^{-15}$ liters, needed to translate concentrations into objects and deterministic rate constant into stochastic ones).

Figure 7 shows the progress of the evolutionary engine while trying to match with a P system the time series generated by the Michaelis-Menten process. Figure 8 shows the actual display of the evolved P system's concentrations and the target concentrations.

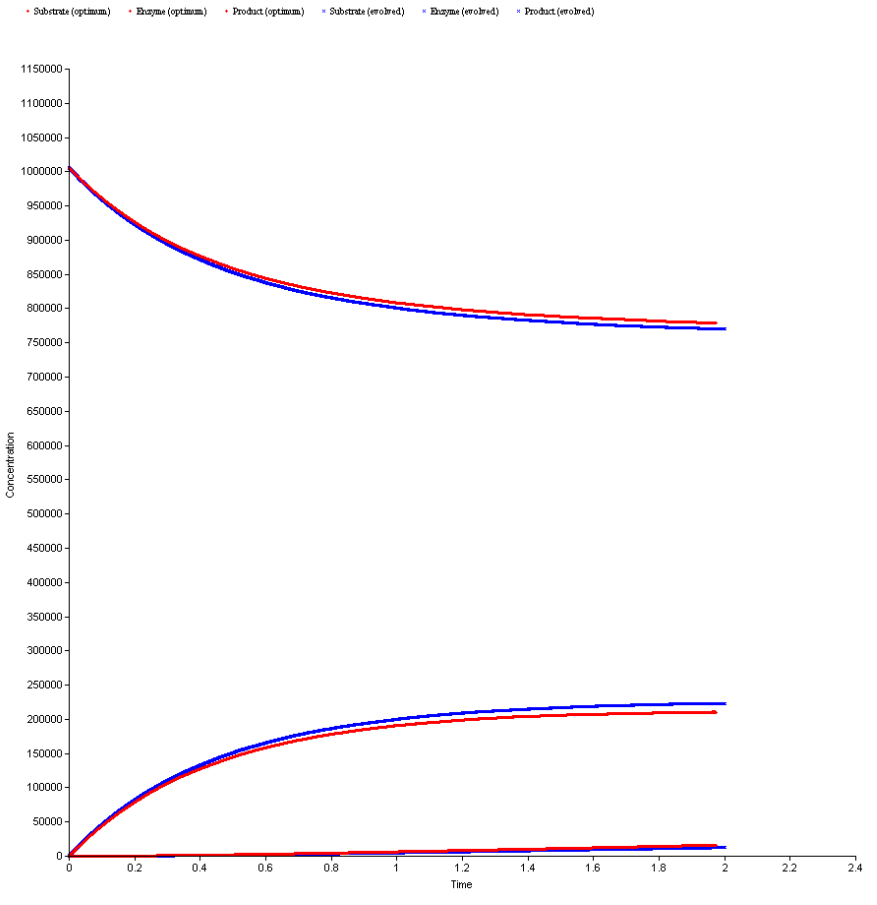


Fig. 8. The target Michaelis-Menten concentrations and the evolved P systems ones

5 Conclusions and Further Work

We have briefly described a part of the quorum sensing network in the Pseudomonas aeruginosa. Starting from a differential equations based model we have provided a P systems version of it and we compared the dynamics of the two approaches. In order to apply different simulation strategies on this intriguing phenomenon we provided a more detailed, mechanistic model which, we believe, is closer to the biological reality. The lack of biological information regarding the dynamics of the system led us to use an automatic way for estimating them by using an evolutionary algorithm approach that offers a reliable and effective method in this respect.

An immediate step further, after obtaining all the parameters regulating a single bacterium dynamics, is to extend the proposed model at a colony level,

exploiting the compartmentalization offered by P systems and already established population P systems models.

Other important developments are related to the use of experimental data to tune the dynamics of our specifications such as to simulate real biological processes. In this respect the use of model checking methodologies, already under consideration in a paper under preparation, will contribute towards validating certain properties of the systems modeled.

On long term we believe that these steps can represent the first stage toward a quantitative analysis that will hopefully lead to a successful drug design process.

Acknowledgements. N. Krasnogor and P. Siepmann acknowledge the EPSRC for funding project EP/D021847/1.

References

1. K. Anguige, J.R. King, J.P. Ward, and P. Williams. Mathematical modelling of therapies targeted at bacterial quorum sensing. *Mathematical Biosciences*, 192:39–83, 2004.
2. A. P. Arkin. Synthetic cell biology. *Current Opinion in Biotechnology*, 12:638–644, 2001.
3. F. Bernardini and M. Gheorghe. Population P systems. *Journal of Universal Computer Science*, 10:509–539, 2004.
4. F. Bernardini, M. Gheorghe, N. Krasnogor, and J.-L. Giavitto. On self-assembly in population P systems. In C.S. Calude, M.J. Dinneen, G. Păun, and M.J. Peŕez-Jimeńez, editors, *Unconventional Computation. 4th International Conference, UC 2005*, pages 46–57, 2005.
5. F. Bernardini, M. Gheorghe, N. Krasnogor, R.C. Muniyandi, M.J. Pérez-Jiménez, and F.J. Romero-Campero. On P Systems as a modelling tool for biological systems. In R. Freund, G. Lojka, M. Oswald, and Gh. Paŭn, editors, *Pre-Proceedings of the 6Th International Workshop on Membrane Computing (WMC6)*, pages 193–213, 2005.
6. L. Bianco, F. Fontana, and V. Manca. P Systems with Reaction Maps. *International Journal of Foundations of Computer Science*, 17(1):27–48, 2006.
7. G.E. Briggs and J.B.S. Haldane. A note on the kinetics of enzyme action. *Biochem. J.*, 19:338–339, 1925.
8. K. A. Connors. *Chemical Kinetics: The study of Reaction Rates in Solution*. VCH, 1990.
9. C.V. Delen and B.H. Iglewski. Cell-to-cell signalling and *Pseudomonas aeruginosa* infections. *Emerging Infectious Diseases*, 4(4):551–560, October-December 1998.
10. S.P. Diggle, K. Winzer, A. Lazdunski, P. Williams, and M. Cámara. Advancing the quorum in *Pseudomonas aeruginosa*: MvaT and the regulation of N-acylhomoserine lactone production and virulence gene expression. *Journal of Bacteriology*, 184:2576–2586, 2002.
11. J.D. Dockery and J.P. Keener. A mathematical model for quorum qensing in *Pseudomonas aeruginosa*. *Bulletin of Mathematical Biology*, 63:95–116, 2001.
12. D.T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22:403–434, 1976.

13. D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Computational Physics*, 81(25):2340–2361, 1977.
14. A.M. Lazdunski, I. Ventre, and J.N. Sturgis. Regulatory circuits and communication in Gram-negative bacteria. *Nature Reviews, Microbiology*, 2:581–592, 2004.
15. G. Păun. Computing with membranes. *J. Comput. System Sci.*, 61(1):108–143, 2000.
16. G. Păun. *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
17. J.P. Pearson. Early activation of quorum sensing. *Journal of Bacteriology*, 184:2569–2571, 2002.
18. M.J. Peŕez-Jimeńez and F. J. Romero-Campero. P systems – A new computational modelling tool for systems biology. *Transactions in Computational Systems Biology*, 2006 (in press).
19. M.J. Pérez-Jiménez and F.J. Romero-Campero. Modelling Vibrio fischeri's behaviour using P systems. In *Systems Biology Workshop, ECAL*, 2005.
20. D. Pescini, D. Besozzi, G. Mauri, and C. Zandron. Dynamical probabilistic P systems. *International Journal of Foundations of Computer Science*, 17(1):183, 2006.
21. P.A. Siepman, G. Terrazas, and N. Krasnogor. Evolutionary Design for the Behaviour of Cellular Automaton-Based Complex Systems. In *Proceedings of the Seventh International Conference on Adapting Computing in Design and Manufacture*.
22. G. Terrazas, N. Krasnogor, M. Gheorghe, F. Bernardini, S. Diggle, and M. Camara. An environment aware P system model of quorum sensing. In S. Barry Cooper, B. Löwe, and L. Torenvliet, editors, *New Computational Paradigms. First Conf. on Computability in Europe, CiE2005*, pages 479–485, 2005.
23. A.U. Viretta and M. Fussenegger. Modelling the quorum sensing regulatory network of human-pathogenic *Pseudomonas aeruginosa*. *Biotechol. Prog.*, 20:670–678, 2004.
24. J.P. Ward, J.R. King, A.J. Koerber, P. Williams, J.M. Croft, and R.E. Sockett. Mathematical modelling of quorum sensing in bacteria. *Journal of Mathematics Applied in Medicine and Biology*, 18:263–292, 2001.
25. K. Winzer, K.R. Hardie, and P. Williams. Bacterial cell-to-cell communication: sorry, can't talk now – gone to lunch! *Current Opinon in Microbiology*, 5:216–222, 2002.

Membrane Systems with External Control

Robert Brijder[1], Matteo Cavaliere[2,3], Agustín Riscos-Núñez[3], Grzegorz Rozenberg[1], and Dragoş Sburlan[3,4]

[1] Leiden Institute of Advanced Computer Science (LIACS)
Universiteit Leiden, Leiden, The Netherlands
rbrijder@liacs.nl, rozenber@liacs.nl
[2] Microsoft Research - University of Trento
Centre for Computational and Systems Biology, Trento, Italy
matteo.cavaliere@msr-unitn.unitn.it
[3] Dept. of Computer Science and Artificial Intelligence
University of Seville, Seville, Spain
ariscosn@us.es
[4] Faculty of Mathematics and Informatics
Ovidius University, Constantza, Romania
dsburlan@univ-ovidius.ro

Abstract. We consider the idea of controlling the evolution of a membrane system. In particular, we investigate a model of membrane systems using promoted rules, where a string of promoters (called the control string) "travels" through the regions, activating the rules of the system. This control string is present in the skin region at the beginning of the computation – one can interpret that it has been inserted in the system before starting the computation – and it is "consumed", symbol by symbol, while traveling through the system. In this way, the inserted string drives the computation of the membrane system by controlling the activation of evolution rules. When the control string is entirely consumed and no rule can be applied anymore, then the system halts – this corresponds to a successful computation. The number of objects present in the output region is the result of such a computation. In this way, using a set of control strings (a control program), one generates a set of numbers. We also consider a more restrictive definition of a successful computation, and then study the corresponding model.

In this paper we investigate the influence of the structure of control programs on the generative power. We demonstrate that different structures yield generative powers ranging from finite to recursively enumerable number sets.

In determining the way that the control string moves through the regions, we consider two possible "strategies of traveling", and prove that they are similar as far as the generative power is concerned.

1 Introduction

Membrane systems (also referred to as P systems) were introduced in 1998 by Gh. Păun as computing devices inspired by the structure and functioning of

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 215–232, 2006.

living cells. Since their introduction, several models of P systems have been investigated, many of them being proved to be computationally complete. The reader is referred to the monograph [6], and to an up-to-date bibliography of this research area available at the P systems web-page, [11].

In nature, the behavior of cells can be influenced by the signals (controls) that they receive from the "outside". Thus, it may be possible to drive the evolution of a living cell by providing the cell with a specific control.

With this motivation in mind, we introduce and investigate a model of P systems, called *string-controlled P systems* (in short, SC P systems). This model is based (with some modifications) on membrane systems with promoters, introduced in [1]. There, the presence of promoters is used to activate, during the computation, certain rules of the system. The biological motivation is the fact that chemical reactions in living cells can be promoted (or inhibited) by the presence of various enzymes.

A string of promoters (called the *control string*), "produced" by the environment, is present in the skin region of the system at the beginning of a computation. This string (that acts like an external control) travels through the regions of the system, possibly promoting (with its leftmost symbol) the rules of the region where it currently resides. Each time the string moves from one region to another, its leftmost symbol (used as a promoter) gets consumed. When the whole string is consumed, and no rule can be applied in any region, then the system halts, completing a successful computation. The output of such computation is the number of objects present in the output region when the system halts.

We shall also consider another sort of successful computation, which additionally has to satisfy a "clean ending condition" (which requires that an a priori specified "undesirable" object is not present in any region upon the completion of the computation).

In this way, an SC P system generates the set of numbers composed by the outputs of all its computations. Also, a membrane system with a collection of control strings (called the *control program*) generates a set of numbers, which is defined as the union of the sets generated for each single string.

In this paper we pay special attention to SC P systems where all evolution rules of the system are promoted – hence, only the rules defined in the region where the control string currently resides, and whose promoter matches the leftmost symbol of the control string, may be active. In particular, we investigate how the structure of the control program influences the generative power of such systems, which are called *fully-promoted SC P systems*.

We show that if the control program is finite, then the generative power corresponds exactly to the family of finite sets of numbers. On the other hand, if the family of recursively enumerable languages is used as the control program, then, not surprisingly, the resulting generative power corresponds to the family of Turing computable sets of numbers. Several intermediate results are obtained by balancing the structure of the control program and the power of the evolution rules used by the system.

We consider two different ways (operating modes) for a control string to travel through the regions of the system: either the string must move at each step (mode (1)), or it is allowed to remain in the same region for several consecutive steps until it decides (nondeterministically) to move again (mode (2)). We prove that, under some natural conditions on the control program, these two modes are similar as far as the generative power is concerned.

The paper is organized as follows. Section 2 recalls some basic notions of formal languages theory used throughout the paper. A formal definition of SC P systems is presented in Section 3. In Section 4 we show that the generative power of classes of fully-promoted SC P systems with a natural condition on the control program family are "almost" independent on the chosen operating mode of the movement of the control string. In Section 5 we consider structures of control that yield a generative power strictly weaker than RE, and in Section 6 structures that yield the computational completeness.

We conclude the paper by suggesting a number of open problems and research directions.

2 Preliminaries

Let us briefly recall some notions and results of formal languages to the extent needed in this paper – in this way we establish the basic notation and terminology needed later on. For more details the reader can consult standard books, such as [10], [2], and the handbook [9].

An *alphabet* V is a finite set of symbols. By V^* we denote the set of all strings over V, the empty string is denoted by λ, and $V^+ = V^* - \{\lambda\}$.

The *length* of a string $w \in V^*$ is denoted by $|w|$, while the number of occurrences of $a \in V$ in w is denoted by $|w|_a$. For a language $L \subseteq V^*$, the set $length(L) = \{|w| \mid w \in L\}$ is called the *length set* of L.

If FL is a family of languages then NFL is the family of length sets of languages in FL.

We denote by FIN, REG, CF, CS and RE the families of finite, regular, context-free, context-sensitive and recursively enumerable languages, respectively. Accordingly, for instance, the family of length sets of languages in RE is denoted by NRE (this is the family of all recursively enumerable sets of natural numbers).

A multiset over V is a mapping $M : V \longrightarrow \mathbb{N}_0$; assigning to each $a \in V$ a multiplicity $M(a)$. Commonly, multisets are represented by strings of symbols. In this representation the order of symbols does not matter, because the number of copies of an object in a multiset is given by the number of occurrences of the corresponding symbol in the string. Hence, e.g., a^4b^3d denotes the multiset consisting of 4 occurrences of a, 3 occurrences of b, and one occurrence of d; the same multiset is also represented by, e.g., $da^2ba^2b^2$.

An ET0L system is a construct $G = (\Sigma, T, H, w)$, where Σ is the (total) alphabet, $T \subseteq \Sigma$ is the terminal alphabet, $H = \{h_1, h_2, \ldots, h_k\}$ is a finite set of finite substitutions (tables) over Σ, and $w \in \Sigma^*$ is the axiom; each $h_i \in H$,

$1 \leq i \leq k$, can be represented by a list of context-free productions $A \to x$, such that $A \in \Sigma$ and $x \in \Sigma^*$ (moreover, for each symbol A of Σ and each table h_i, $1 \leq i \leq k$, there is a production in h_i with A as the left hand side). Then G defines, for each $1 \leq i \leq k$, a derivation relation $\Rightarrow_{h_i}$ by $x \Rightarrow_{h_i} y$ iff $y \in h_i(x)$. We write $x \Rightarrow y$ if $x \Rightarrow_{h_i} y$ for some $1 \leq i \leq k$. As usual, $x \Longrightarrow^* y$ denotes the reflexive and transitive closure.

The language generated by G is $L(G) = \{z \in T^* \mid w \Longrightarrow^* z\}$. We denote by $ET0L$ the family of languages generated by ET0L systems, and by $T0L$ the family of languages generated by ET0L systems such that $\Sigma = T$.

A regularly (context-free, respectively) controlled ET0L system, in short E(rc) T0L system (E(cfc)T0L system, respectively), is a pair $\Omega = (G, L)$ where $G = (\Sigma, T, H, w)$ is an ET0L system and L is a regular (context-free, respectively) language over H.

The language generated by Ω is

$$L(\Omega) = \{z \in T^* \mid w = w_0 \Rightarrow_{h_{i_1}} w_1 \Rightarrow_{h_{i_2}} \ldots \Rightarrow_{h_{i_m}} w_m = z,\ h_{i_1} \cdots h_{i_m} \in L\}.$$

We denote by $E(rc)T0L$ the family of languages generated by E(rc)T0L systems, and by $E(cfc)T0L$ the family of languages generated by E(cfc)T0L systems.

The following known inclusions between families of languages will be used in this paper (see, e.g., [10]):

$$FIN \subset CF \subset ET0L \subset CS \subset RE.$$

From [4] we recall the following result.

$$ET0L = E(rc)T0L.$$

Moreover, it is known that for each $L \in ET0L$ there exists an ET0L system G, with only 2 tables, such that $L = L(G)$ (see, e.g., [8]).

A *regularly controlled grammar with appearance checking* is a tuple $G = (N, T, S, P, K, F)$ where N, T, S, and P are the set of nonterminals, the set of terminals, the starting symbol and a finite set of context-free productions, respectively. Each production in P has a uniquely associated label, and the set of all these labels is denoted by $lab(P)$. K is a regular language over $lab(P)$ and $F \subseteq lab(P)$. Let $V = N \cup T$. We say that $x \in V^+$ *derives* $y \in V^*$ in the appearance checking mode by application of $A \to w$ with label p (written as $x \Rightarrow_p^{ac} y$) if either $x = x_1 A x_2$ and $y = x_1 w x_2$, or A does not appear in x, $p \in F$, and $x = y$.

The language $L(G)$, generated by G, consists of all strings $w \in T^*$ such that there is a derivation $S \Rightarrow_{p_{i_1}}^{ac} w_1 \Rightarrow_{p_{i_2}}^{ac} w_2 \Rightarrow_{p_{i_3}}^{ac} \ldots \Rightarrow_{p_{i_n}}^{ac} w_n = w$, for some $n \geq 1$ and $p_{i_1} p_{i_2} \cdots p_{i_n} \in K$.

By rC_{ac} we denote the family of languages generated by regularly controlled grammars with appearance checking and erasing productions, and by rC we denote the family of languages generated by regularly controlled grammars with erasing productions and without appearance checking (the set F is empty).

The following lemma holds (see [2]):

Lemma 1. $rC_{ac} = RE$.

In what follows we assume that the reader is familiar with the membrane computing area, in particular with the class of P systems with rewriting rules and symbol-objects, and with the notions of P systems using promoters/inhibitors; for instance as presented in [1,5,7] or in Chapter 3 of [6].

3 String-Controlled P Systems

A string-controlled P system, as informally described in Introduction, is defined as follows.

Definition 1. *A string-controlled P system (in short, SC P system) is a construct*

$$\Pi = (V, C, P, L, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0),$$

where:

- V *is the alphabet of* Π*; its elements are called* objects*;*
- $C \subseteq V$ *is the set of* catalysts*;*
- P *is the set of* promoters*;* $P \cap V = \emptyset$*;*
- $L \subseteq P^*$ *is the* control program *(each string in* L *is a* control string*);*
- μ *is a* membrane structure *consisting of* m *membranes labeled* $1, \ldots, m$*;*
- w_i, $1 \leq i \leq m$, *are strings that represent the multisets over* V *initially associated with the regions* $1, 2, \ldots, m$ *of* μ*;*
- R_i, $1 \leq i \leq m$, *are finite sets of* evolution rules *associated with the regions* $1, 2, \ldots, m$ *of* μ*. Each evolution rule is either of the form* $u \to v$ *or of the form* $u \to v|_p$*, where* $u \in V^+$, $p \in P$, *and* $v \in V_{tar}^*$ *with* $V_{tar} = V \times TAR$*, for* $TAR = \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$*;*
- $i_0 \in \{1, \ldots, m\}$ *specifies the output region of* Π*.*

As usual, the *membrane structure* is a hierarchical arrangement of membranes, embedded in a *skin membrane*, which separates the system from the environment. A membrane without any membrane inside is called *elementary*. Each membrane defines a *region*. For an elementary membrane this is the space enclosed by it, while for a non-elementary membrane, is the space in-between the membrane and the membranes directly included in it. As usual, labels $1, \ldots, m$ identify both membranes and their corresponding regions.

Evolution rules of the form $u \to v|_p$ are called *promoted*, and evolution rules of the form $u \to v$ are called *non-promoted*. An evolution rule is called *non-cooperative* if $u \in V$. Also, an evolution rule is called *catalytic* if it is either of the form $ca \to cv$ or of the form $ca \to cv|_p$, where $a \in (V - C)$, $c \in C$, $p \in P$, and $v \in \big((V - C) \times TAR\big)^*$. The elements of TAR are called *targets*. It is convenient to denote $(a, t) \in V_{tar}$ by a if $t = here$, and by a_t otherwise.

A *configuration* of Π is a description of the membrane structure and of the contents of all the regions. An *initial configuration* of Π consists of the membrane

structure μ, the objects initially present in the regions of the system, as described by $w_1, \ldots, w_m$, and by one string from L, present in the skin region (this string is called *control string*). Notice that Π has a set of initial configurations, one for each element of L.

As standard, we suppose the existence of a global clock that marks the steps of the system.

At each step, the control string moves, in a nondeterministic way, across the regions of Π. We distinguish *two possible modes* of operation for Π: (1) at each step the string moves passing from one region to an adjacent one; (2) at each step the string may move to an adjacent region or remain in the same region. In both cases the control string cannot move to the environment, and when it moves from a region to another one, it loses its leftmost symbol. The leftmost symbol of the control string is called the *head*.

At each step the *head* of the current control string is used as a *promoter* for the rules present in the region where the string resides. A promoted rule is *active* if its promoter is present. The rules that are not promoted are always active.

A *transition* between two configurations of Π is obtained by applying in one step the active rules in each region of Π in a maximally parallel nondeterministic manner. More precisely, if a rule $u \to v \in R_i$ or $u \to v|_p \in R_i$ is active and the multiset u is present in region i, then the *application* of this rule means removing u from region i and adding the objects specified by v in the regions indicated by the corresponding target commands.

A sequence of transitions, starting from an initial configuration of Π, is called *computation*. A computation *halts* when there is no applicable rule in any region of Π *and* the control string is entirely consumed (Π has reached a *halting configuration*).

We shall consider two definitions of *successful* computation for Π:

- in the standard case, we say that all halting computations of Π are successful,
- in the # case, we consider that a halting computation of Π is successful if and only if a special *a priori* designated symbol $\# \in V$ is not present in the halting configuration in any region of Π.

The *result* of a successful computation ω is the number of objects present in the output region i_0 in the halting configuration of ω. Depending on the definition of *successful* computation that is considered, we shall say that the system collects the result in the standard way, or in the # way.

We use the notation $P_m(\alpha, FL)$, where $\alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 1\}$ and FL is a family of languages, to denote the class of SC P systems which use at most m membranes, use only non-cooperative ($ncoo$), cooperative (coo), or catalytic with at most k catalysts (cat_k) evolution rules (promoted or not), and use a control program in FL. We call FL the *control program family* of the class. In the coo case, there is no restriction on the form of the evolution rules. The prefix (pro) is added if *only* promoted rules are used (such systems are called *fully-promoted* SC P systems).

We denote by $N^{(i)}(\Pi), i \in \{1, 2\}$, the set of results of all successful computations of Π starting from any possible initial configuration, operating in mode (i),

and collecting the result in the standard way. Similarly, we denote by $N_{\#}^{(i)}(\Pi), i \in \{1,2\}$, the set of results of all successful computations of Π operating in mode (i) and collecting the result in the # way. Moreover, $N^{(i)}P_m(\alpha, FL) = \{N^{(i)}(\Pi) \mid \Pi \in P_m(\alpha, FL), i \in \{1,2\}\}$ denotes the family of sets of natural numbers generated by SC P systems from $P_m(\alpha, FL)$ operating in mode (i), $i = 1, 2$, and collecting the result in the standard way. The family $N_{\#}^{(i)}P_m(\alpha, FL)$ is similarly defined.

The following inclusions follow directly from the definitions.

Lemma 2

$$(pro)N^{(i)}P_m(\alpha, FL) \subseteq (pro)N_{\#}^{(i)}P_m(\alpha, FL),$$
$$(pro)N_{\#}^{(i)}P_m(\alpha, FL_1) \subseteq (pro)N_{\#}^{(i)}P_m(\alpha, FL_2), \text{ if } FL_1 \subseteq FL_2,$$

$$(pro)N_{\#}^{(i)}P_m(ncoo, FL) \subseteq (pro)N_{\#}^{(i)}P_m(cat_j, FL)$$
$$\subseteq (pro)N_{\#}^{(i)}P_m(cat_{j+1}, FL) \subseteq (pro)N_{\#}^{(i)}P_m(coo, FL),$$

for $j \geq 1$, $i \in \{1,2\}$, $\alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 1\}$, *and* FL, FL_1, FL_2 *families of languages.*

4 Fully-Promoted SC P Systems

In this section we start the investigation of fully-promoted SC P systems. Notice that for such systems in each time step there is activity in at most one region (the region where the control string currently resides). First we give an example that illustrates the functioning of an SC P system. Then we prove the equivalence (as far as the generative power is concerned) between modes (1) and (2).

The following example shows that a given SC P system Π can produce different results according to its functioning mode.

Example 1. Let Π be the SC P system:

$$\Pi = (V, C, P, L, \mu, w_1, w_2, R_1, R_2, i_0),$$

where:

- $V = \{A\}$,
- $C = \emptyset$,
- $P = \{a, b\}$,
- $L = \{ab\}$,
- $\mu = [_1\, [_2\ ]_2\,]_1$,
- $w_1 = \lambda; w_2 = A$,
- $R_1 = \emptyset$,
- $R_2 = \{A \to AA|_b\}$,
- $i_0 = 2$.

The system collects the result in the standard way.

When Π operates in mode (1), the unique control string of L is initially present in the skin region and moves, in the next step, to region 2, losing its head a. Therefore, now the rule $A \to AA|_b$ is activated. In the following step the control string exits region 2, entering region 1, and then its last symbol, b, is consumed. Therefore, there is only one successful computation and we have $N^{(1)}(\Pi) = \{2\}$.

If Π operates in mode (2), then the unique control string of L is initially present in the skin region and it may remain there for a certain number of steps; meanwhile nothing is produced in region 2. At a certain step the string moves into region 2, losing its head a. Then, the rule $A \to AA|_b$ is activated in region 2 and it will double the number of objects A at each step, until the string b moves back to region 1. When this happens the computation halts and the number of objects produced in region 2 is a power of two, that is, $N^{(2)}(\Pi) = \{2^n \mid n \geq 1\}$.

Example 1 illustrates that for a given fully-promoted SC P system the generated sets under operating modes (1) and (2) may differ (even drastically). However, the family of sets of numbers generated by a *class* of fully-promoted SC P systems with a control program family that is closed under non-erasing regular substitution is "almost" independent on the chosen operating mode. In fact, we show that any fully-promoted SC P system operating in mode (2) [(1), respectively] can be simulated (in a weak sense) by a fully-promoted SC P system operating in mode (1) [(2), respectively] using the same type of rules, the same type of control program, and using a double number of membranes.

Theorem 1. *Let $\Pi \in (pro)P_m(\alpha, FL)$, where $m \geq 1$, $\alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 1\}$, and FL is closed under non-erasing regular substitution. There exists $\Pi' \in (pro)P_{2m}(\alpha, FL)$, such that*

$$N^{(1)}_{\#}(\Pi') = \{x + 1 \mid x \in N^{(2)}_{\#}(\Pi)\}.$$

Proof. Let $\Pi = (V, C, P, L, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0) \in (pro)P_m(\alpha, FL)$, and let us construct $\Pi' = (V', C, P', L', \mu', w'_1, \ldots, w'_{2m}, R'_1, \ldots, R'_{2m}, i_0) \in (pro)P_{2m}(\alpha, FL)$ as follows.

Let $V' = V \cup \{Z\}$, with $Z \notin V$ and $P' = P \cup \{d\}$, with $d \notin P$. We consider the regular substitution ϕ defined by $\phi(p) = p(dp)^*$ for each $p \in P$; we define $L' = \phi(L)$ (notice that the substitution is non-erasing and so every family of languages in $\{REG, CF, CS, RE\}$ is closed under this operation). The structure μ' has $2m$ membranes and is obtained from μ by adding, in each region $i, 1 \leq i \leq m$, of μ an (elementary) membrane with label $m + i$. Furthermore we define $w'_i = w_i Z$, for $1 \leq i \leq m$, and $w'_i = Z$, for $m + 1 \leq i \leq 2m$.

We define $R'_i = R_i \cup \{Z \to \#|_d\}$, for $1 \leq i \leq m$, and $R'_i = \{Z \to \#|_p \mid p \in P\}$, for $m + 1 \leq i \leq 2m$.

We shall now show that for every successful computation $\mathcal{C}$ of Π with result x operating in mode (2) there exists a successful computation $\mathcal{C}'$ of Π' with result $x + 1$ operating in mode (1).

Consider an arbitrary computation of Π and consider one of its configurations. Now, suppose that in such configuration the current control string w is in region i

of Π and has p as its head. Then, there exists a computation in Π', starting with an "appropriate" control string from L' in the skin, that reaches a configuration having the control string $w' = p(dp)^n x$ present in region i of Π'.

Suppose now that w does not move in Π (Π operates in mode (2)) but remains in the same region for several consecutive steps. This is simulated in Π' by moving w' back and forth between region i and the adjacent dummy region $m+i$, consuming for each movement a symbol p and a dummy promoter d. In this way, an arbitrary computation in Π can be simulated in Π' by a computation starting with an appropriate control string from L'.

On the other hand, Π' does not have other successful computations except those simulating successful computations of Π as described above. In fact, since there is a rule $Z \to \#|_d$ in every set R'_i, for $1 \leq i \leq m$, which guarantees that the dummy symbol d cannot be used to move the control string into non-dummy regions, otherwise the computation would not be successful. Moreover, if the promoter present immediately to the right of the head of the current control string is non-dummy (i.e., the string is of type pqx, with $p, q \in P$, and $x \in P^*$), then the string must move in a non-dummy region, because otherwise the rules $R'_i = \{Z \to \#|_p \mid p \in P\}$, for $m+1 \leq i \leq 2m$, would make the computation unsuccessful, if applied. From the above discussion it should be clear that the theorem holds. □

Conversely, a fully-promoted SC P system operating in mode (1) can be simulated (in a weak sense) by a fully-promoted SC P system operating in mode (2), using a structure having a double number of membranes.

Theorem 2. *Let $\Pi \in (pro)P_m(\alpha, FL)$, where $m \geq 1$, $\alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 1\}$, and FL is closed under non-erasing morphism. There exists $\Pi' \in (pro)P_{2m}(\alpha, FL)$, such that*

$$N^{(2)}_{\#}(\Pi') = \{x + 2 \mid x \in N^{(1)}_{\#}(\Pi)\}.$$

Proof. Given $\Pi = (V, C, P, L, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0)$ we construct $\Pi' = (V', C, P', L', \mu', w'_1, \ldots, w'_{2m}, R'_1, \ldots, R'_{2m}, i_0)$ as follows.

Let $V' = V \cup \{c, c', Z\}$ and $P' = P \cup \{d, d'\}$, with $c, c', Z \notin V$, and $d, d' \notin P$. We consider the non-erasing morphism ϕ defined by $\phi(p) = pdd'$, for each $p \in P$ – then we set $L' = \phi(L)$ (notice that every family of languages in $\{FIN, REG, CF, CS, RE\}$ is closed under non-erasing morphisms). The membrane structure μ' has $2m$ membranes and is obtained from μ in the following way. In each region $i, 1 \leq i \leq m$, of μ an (elementary) membrane with label $m+i$ is added.

The initial multisets of Π' are $w'_i = cZw_i$, for $1 \leq i \leq m$, and $w'_i = Z$, for $m+1 \leq i \leq 2m$.

Finally, the evolution rules of Π' are defined in the following way: $R'_i = R_i \cup \{c' \to c|_{d'}, Z \to \#|_d\} \cup \{c' \to \#|_p, c \to c'|_p \mid p \in P\}$, for $1 \leq i \leq m$. $R'_i = \{Z \to \#|_p \mid p \in P\}$, for $m+1 \leq i \leq 2m$.

We will prove now that for every computation of Π operating in mode (1) and producing x, there exists a computation of Π' operating in mode (2) producing $x+2$.

Consider an arbitrary computation of Π and suppose that, after a certain step k during that computation, the control string $p_{i_1}p_{i_2}\cdots p_{i_j}$, with $p_{i_1}, p_{i_2}, \ldots, p_{i_j} \in P$, is present in region i of Π.

Then, there is a computation of Π' (starting with an "appropriate" control string from L') such that the control string $p_{i_1}dd'p_{i_2}dd'\cdots p_{i_j}dd'$ is present in region i of Π' after a given step k'.

In Π, at step $k+1$, the string must exit region i (Π operates in mode (1)), entering one of the adjacent regions, chosen nondeterministically, losing the promoter p_{i_1} and getting the promoter p_{i_2} as its new head.

This single step of Π is simulated by Π' in the following consecutive steps. The rules activated by promoter p_{i_1} present in region i of Π' are executed at step k', together with the rule $c \to c'$ present in every region of Π' and activated by any promoter of P. Therefore, at step $k'+1$ the control string must exit region i, as otherwise in the next step the rule $c' \to \#|_{p_{i_1}}$ would be applied and the entire computation would not be successful.

The only region of Π' where the control string can go to is the dummy region $m+i$ present inside region i (otherwise the promoter d that follows p_{i_1} would activate the rule $Z \to \#|_d$ present in any of the non-dummy adjacent regions of region i and the computation would not be successful). Therefore, suppose the control string goes to region $m+i$, losing in this way the promoter p_{i_1}; the control string may remain in region $m+i$ for an unbounded number of steps (no rule can be applied there). At a certain step k'' the control string comes back to region i, losing the promoter d and having now the promoter d' as its head; therefore, in the step $k''+1$ the rule $c' \to c|_{d'}$ is applied. The control string having now d' as head may remain in region i for an unbounded number of steps (no rule can be applied). Eventually, the control string exits region i moving to an adjacent region, losing the promoter d', and having the promoter p_{i_2} (the next non-dummy promoter) as its new head.

Thus, all possible movements of the control string in Π (i.e., all possible computations) are correctly captured by the functioning of Π'; consequently, every successful computation of Π can be simulated by Π'.

Notice that, in Π', if the promoter adjacent to the head of the control string is non-dummy (i.e., it belongs to the set P), then the control string must move in a non-dummy region; otherwise a rule from $R'_i = \{Z \to \#|_p \mid p \in P\}$, $m+1 \leq i \leq 2m$, is applied and that would make the computation unsuccessful.

Therefore there are no other successful computations of Π' except those that simulate, in the above described way, successful computations of Π. Thus, the theorem holds. □

5 The Influence of the Control Program

Now we analyze in more detail the class of fully-promoted SC P systems operating in mode (1). We show how the structure of the control program and the type of evolution rules influence the generative power of the constructed

membrane system. A series of results, ranging from finite power to computational universality, is obtained.

It is worth to remark that one can easily obtain the length set of any language L as output of an SC P system using non-cooperative rules and having L as the control program. Hence, the structure of the control program influences the generative power of SC P systems as the following theorem states.

Theorem 3. $NFL \subseteq (pro)N^{(1)}P_2(ncoo, FL)$.

Proof. Given an arbitrary language L over the alphabet $\Sigma = \{a_1, \ldots, a_n\}$, let us consider a symbol $* \notin \Sigma$, and let $L' = h(L)$ where h is the morphism defined by $h(a) = *a$, for every $a \in \Sigma$.

Now let us construct an SC P system that generates $length(L)$ as follows:

$$\Pi = (V, C, P, L', \mu, w_1, w_2, R_1, R_2, i_0),$$

where:

- $V = \{a'_1, \ldots, a'_n\}$,
- $C = \emptyset$,
- $P = \Sigma$,
- $\mu = [_1\, [_2\ \ ]_2\,]_1$,
- $w_1 = \lambda; w_2 = a'$,
- $R_1 = \emptyset$,
- $R_2 = \{a' \to a'_{out} a'|_a \mid a \in \Sigma\}$,
- $i_0 = 1$.

At the beginning of the computation one of the strings from L', nondeterministically chosen, is present in the skin region of Π (i.e., region 1). The string moves back and forth between region 1 and region 2 of the system, losing alternatively the symbol $*$ (when passing from region 1 to region 2) and a symbol $a \in \Sigma$ (when moving in the opposite direction). When the string is in region 2, its head $a \in \Sigma$ activates exactly the rule that produces and sends out the symbol a'. Therefore, the number of symbols contained in the output region when the computation halts (the string is entirely consumed) is equal to the number of symbols from Σ that occurred in the inserted control string. Thus Π generates exactly the $length(L)$. □

Now, from Corollary 2, Theorem 3 and the Turing-Church thesis, we have that the class of fully-promoted SC P systems using arbitrary RE languages as control program is universal, even when only non-cooperative rules are used. Hence, the following theorem holds.

Theorem 4. $(pro)N^{(1)}P_2(ncoo, RE) = (pro)N^{(1)}_{\#}P_2(ncoo, RE) = NRE$.

It is now natural to ask what happens if we increase the "power" of the evolution rules used by the P system and we decrease the "power" of the control program.

First we consider SC P systems that use cooperative evolution rules and finite control programs.

Theorem 5. $(pro)N^{(1)}_{\#}P_*(coo, FIN) = (pro)N^{(1)}P_*(coo, FIN) = NFIN.$

Proof. Given an SC P system Π, it is sufficient to notice that the number of distinct nondeterministic computations using only a finite number of steps is bounded by a constant that only depends on Π. Therefore, if Π has a finite control program, then the set of numbers produced is finite. The other inclusion follows from Theorem 3. □

Let us prove next that the class of fully-promoted SC P systems using arbitrary context-free (regular, respectively) languages as control program generates exactly the family $NE(cfc)T0L$ (or the family $NE(rc)T0L$, respectively), even with non-cooperative rules.

Theorem 6

$$(pro)N^{(1)}_{\#}P_2(ncoo, REG) \supseteq NE(rc)T0L = NET0L.$$

$$(pro)N^{(1)}_{\#}P_2(ncoo, CF) \supseteq NE(cfc)T0L.$$

Proof. Given $\Omega = (G, L)$ an arbitrary E(rc)T0L system (or E(cfc)T0L system, respectively) we construct a SC P system Π in $(pro)P_2(ncoo, REG)$ (in $(pro)P_2(ncoo, CF)$, respectively) such that $N^{(1)}_{\#}(\Pi) = length(L(\Omega))$ as follows.

Let $G = (\Sigma, T, H, w)$ with $H = \{h_1, \ldots, h_k\}$. Let

$$\Pi = (V, C, P, L, \mu, w_1, w_2, R_1, R_2, i_0),$$

where:

- $V = \Sigma$,
- $C = \emptyset$,
- $P = \{t_1, \ldots, t_k, d, p\}$, with $d, p \notin \{t_1, \ldots, t_k\}$,
- $L' = \phi(L)dp$ with the morphism ϕ defined by $\phi(t_i) = dt_i, 1 \leq i \leq k$,
- $\mu = [_1\, [_2\ \ ]_2\,]_1$,
- $w_1 = \lambda; w_2 = w$,
- $R_1 = \emptyset$,
- $R_2 = \{X \to \alpha|_{t_i} \mid X \to \alpha \in h_i, 1 \leq i \leq k\} \cup \{N \to \#|_p \mid N \in \Sigma - T\}$,
- $i_0 = 2$.

Now Π simulates in region 2 the productions of G, applying the tables according to the strings in L', in such a way that each table h_i has an associated promoter t_i, for every $1 \leq i \leq k$.

The dummy promoter d is only used to be consumed while the control string moves from region 1 to region 2. In this way, the new head of the control string is a symbol t_i, for some $1 \leq i \leq k$. The final promoter p added as last symbol of any string in L' is used to check whether or not there are still nonterminals in region 2 in the last step of the computation. If this is the case, then the special object # is produced and the computation is not successful. Consequently, the theorem follows. □

We continue now to prove that the reverse inclusions also hold.

Theorem 7

$$(pro)N_{\#}^{(1)}P_*(ncoo, REG) \subseteq NE(rc)T0L = NET0L.$$
$$(pro)N_{\#}^{(1)}P_*(ncoo, CF) \subseteq NE(cfc)T0L.$$

Proof. Consider a fully-promoted SC P system Π of the form

$$\Pi = (V, C, P, L, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0),$$

such that $C = \emptyset$ and L is a regular (context-free, respectively) language over $P = \{p_1, p_2, \ldots, p_k\}$.

We consider the morphisms φ_i, $1 \leq i \leq m$, defined by $\varphi_i(X) = (X, i)$, for all $X \in V$, $1 \leq i \leq m$. By using these morphisms, we associate with each occurrence of any object X the index of the region where the occurrence resides.

We also use the morphisms φ_i^t, $1 \leq i \leq m$, defined by

$$\varphi_i^t(X_{tar}) = \begin{cases} (X, i) & \text{if } tar = here, \\ (X, j) & \text{if } tar = out, \\ (X, k) & \text{if } tar = in_k, \end{cases}$$

for all $X \in V$, where j is the label of the surrounding region of i.

We construct now an E(rc)T0L system (or an E(cfc)T0L system, respectively) $\Omega = (G, L')$ simulating the computations of Π.

First we construct G. Let $G = (\Sigma, T, H, w')$, where $\Sigma = \{(X, i) \mid X \in V, 1 \leq i \leq m\}$, $T = \Sigma - \{(\#, i) \mid 1 \leq i \leq m\}$ and $w' = \varphi_1(w_1) \cdots \varphi_m(w_m)$.

Each table $h_{i,p_j} \in H$, $1 \leq i \leq m$, $1 \leq j \leq k$, is constructed in the following way:

- for each $X \in V$, if $X \to \alpha|_{p_j} \in R_i$, for some $p_j \in P$, then the rule $(X, i) \to \varphi_i^t(\alpha)$ is added to the table h_i. Otherwise, if X is not present as the left hand side of any rule in R_i, then the rule $(X, i) \to (X, i)$ is added to the table h_i;
- for each $X \in V$ and $1 \leq l \leq m$, $l \neq i$, the rule $(X, l) \to (X, l)$ is added to the table h_i.

Notice that H has mk tables and each one of them is complete.

Finally we construct L'. To this aim we define the finite substitution φ' by $\varphi'(p_j) = \{t_{(i,p_j)} \mid 1 \leq i \leq m\}$ for each $1 \leq j \leq k$. We also define the nondeterministic finite state automaton $A = (Q, V_A, s_0, F, \delta)$, where $Q = \{0, 1, \ldots, m\}$, $V_A = \{t_{(i,p_j)} \mid 1 \leq i \leq m,\ 1 \leq j \leq k\}$, $s_0 = 0$, $F = Q$ and δ is defined by $\delta(0, t_{(1,p_j)}) = 1$, $\delta(i_1, t_{(i_1,p_j)}) = \{i_2 \mid 1 \leq i_2 \leq m$, and region i_1 is adjacent to region i_2 in $\mu\}$ for every $1 \leq j \leq k$ and $1 \leq i_1 \leq m$. Without loss of generality, we assume 1 to be the label of the skin membrane of Π.

Now, $L' = \varphi'(L) \cap L(A)$ is regular (context-free, respectively) since regular (context-free, respectively) languages are closed under intersection with regular languages, see e.g. [10].

The underlying idea of the proof is the following.

Each table $t_{(i,p_j)}$ of G with $1 \leq i \leq m, 1 \leq j \leq k$, simulates the rewriting in parallel of the objects present in region i of Π, by using rules activated by the

promoter p_j. All the objects present in the same region that cannot be rewritten by any active rule, as well as those present in the other regions of the system, are left unchanged by the application of the table.

The language $\varphi'(L)$ is used to pass from one table to another, in the way described by the strings of promoters present in the control program L. More specifically, if the string $w = p_{j_1} \cdots p_{j_l}$ is present in L, then $\varphi'(L)$ contains all the strings of the set $S_w = \{t_{(i_1,p_{j_1})}, \ldots, t_{(i_l,p_{j_l})} \mid i_1, \ldots, i_l \in \{1, \ldots, m\}\}$. In this way, each computation of Π starting with the control string $w = p_{j_1} \cdots p_{j_l}$ can be simulated in G by applying the tables following the order of an appropriate string in S_w. On the other hand, not every string in the set S_w simulates a correct computation in Π starting with the control string $p_{j_1} \cdots p_{j_l}$. In fact, the control string in Π can only move through adjacent regions – this has to be "encoded" in the way that the passage from one table of G to another one is done. For this reason the appropriate regular (context-free, respectively) language L' that controls G is obtained by intersecting the language $\varphi'(L)$ with the regular language $L(A)$.

From the above explanation it follows that each string in $L(\Omega)$ contains pairs $(object, region)$ corresponding to the objects present in the halting configurations of successful computations of Π. In order to get the exact contents of the output region of Π, we apply to $L(\Omega)$ the morphism φ_o, defined by:

$$\varphi_o((X,i)) = \begin{cases} X & \text{if } i = i_0, \\ \lambda & \text{otherwise.} \end{cases}$$

Since the family $E(rc)T0L$ (or the family $E(cfc)T0L$, respectively) is clearly closed under arbitrary morphisms, it follows that $N_{\#}^{(1)}(\Pi)$ belongs to the family $NE(rc)T0L$ (or to the family $NE(cfc)T0L$, respectively). Thus the theorem holds. □

From Theorems 6 and 7 we obtain

Corollary 1

$$(pro)N_{\#}^{(1)}P_*(ncoo, REG) = NE(rc)T0L = NET0L.$$
$$(pro)N_{\#}^{(1)}P_*(ncoo, CF) = NE(cfc)T0L.$$

On the other hand, if SC P systems collect the result in the standard way, then one gets the following results.

Theorem 8

$$(pro)N^{(1)}P_2(ncoo, REG) \supseteq N(rc)T0L = NET0L.$$
$$(pro)N^{(1)}P_2(ncoo, CF) \supseteq N(cfc)T0L.$$

Proof. In the proof of Theorem 6 the special symbol # is only used to check if any nonterminal of G is still present when the computation of Π halts. Therefore this checking can be avoided during the simulation of a (rc)T0L system (or a (cfc)T0L system, respectively). Hence the theorem holds. □

Analogously, note that in Theorem 7 the set of nonterminals used by the ET0L system constructed in the proof contains only the special object # included in the alphabet of the corresponding SC P system Π. Therefore if Π collects the output in the standard mode (i.e., it does not use #), then one gets the following results.

Theorem 9

$$(pro)N^{(1)}P_*(ncoo, REG) \subseteq N(rc)T0L = NET0L.$$
$$(pro)N^{(1)}P_*(ncoo, CF) \subseteq N(cfc)T0L.$$

Theorems 8 and 9 yield the following corollary.

Corollary 2

$$N(rc)T0L = (pro)N^{(1)}P_*(ncoo, REG) = NET0L.$$
$$N(cfc)T0L = (pro)N^{(1)}P_*(ncoo, CF).$$

6 Fully-Promoted SC P Systems: Universality

If SC P systems use arbitrary regular control programs, and only one catalyst, then they generate the family of recursively enumerable sets of natural numbers.

In [3], P systems using two catalysts and two membranes have been proved to be universal. This proof can also be applied for non fully-promoted SC P systems to obtain the following universality result.

Corollary 3. $N_{\#}^{(1)}P_2(cat_2, \{\{\lambda\}\}) = NRE.$

In case of fully-promoted SC P systems, the computational universality can be obtained using arbitrary regular control programs and catalytic rules with only one catalyst.

Theorem 10. $(pro)N_{\#}^{(1)}P_2(cat_1, REG) = NRE.$

Proof. The inclusion in NRE follows from Church-Turing thesis. The opposite inclusion can be proved by simulating regularly controlled grammars with appearance checking, as follows.

Given a regularly controlled grammar with appearance checking $G = (N, T, S, P, K, F)$, we construct $\Pi \in (pro)P_2^{(1)}(cat_1, REG)$, collecting the output in the # way, that simulates G. Let

$$\Pi = (V, C, P', L, \mu, w_1, w_2, R_1, R_2, i_0),$$

where:

- $V = N \cup T \cup \{c, Z\}$,
- $C = \{c\}$,
- $P' = lab(P) \cup \{d, d'\}, d, d' \notin lab(P)$,

- $L = \phi(K)dd'$ with non-erasing morphism ϕ defined by $\phi(p) = dp$ for each $p \in lab(P)$,
- $\mu = [_1\, [_2\ ]_2\,]_1$,
- $w_1 = \lambda; w_2 = SZc$,
- $R_1 = \emptyset$,
- $R_2 = \{cA \to c\alpha|_p \mid p : A \to \alpha \in P\} \cup \{cZ \to c\#|_p \mid p \notin F\}$ $\cup \{Z \to Z_{out}|_{d'}\}$,
- $i_0 = 2$.

We show that Π simulates the derivations of G. Note that, by definition, for every $p_{i_1} \cdots p_{i_k} \in K$, we have $dp_{i_1} \cdots dp_{i_k} dd' \in L$. The promoters d are dummies, they are only used to let the control string to enter and exit region 2, passing in this way from a promoter p_{i_j} as the current head to the promotor $p_{i_{j+1}}$, for $1 \leq j \leq k-1$. The derivations of G are simulated by the execution of rules from R_2.

Notice that, because of the catalyst c that inhibits the parallelism, at most one rule is executed in region 2 when the control string resides in that region. If a rule cannot be applied and the label of the corresponding production is not in F, then the computation is unsuccessful (# is produced by applying the rule $cZ \to c\#$ that is activated by any promoter $p \in (lab(P) - F)$) and this is correct since the simulated derivation in G cannot be continued. On the other hand, if a rule cannot be applied and the label of the corresponding production is in F (so the production has to be used in the appearance checking mode), then no rule is applied in region 2, the control string leaves the region and the computation continues. The last promoter d' present for any control string in L is used to move, at the end of the computation, the symbol Z into region 1. It should be clear from the above description that $N_{\#}^{(1)}(\Pi)$ is exactly the length set of $L(G)$. Thus the theorem holds. □

We conclude this section by presenting some preliminary results concerning the class of non fully-promoted SC P systems.

By definition, it is clear that

Lemma 3

$$(pro)N_{\#}^{(i)}P_m(\alpha, FL) \subseteq N_{\#}^{(i)}P_m(\alpha, FL),$$
$$(pro)N^{(i)}P_m(\alpha, FL) \subseteq N^{(i)}P_m(\alpha, FL),$$

for $\alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 1\}$, FL *a family of languages, and* $i \in \{1, 2\}$.

It is easy to notice that systems from $P_1(ncoo, FIN)$ can generate infinite sets of numbers when operating in mode (1) and collecting the result in the standard way. This observation and Theorem 5 yield the following result.

Theorem 11

$$(pro)N_{\#}^{(1)}P_*(\alpha, FIN) = (pro)N^{(1)}P_*(\alpha, FIN) \subset N^{(1)}P_*(\alpha, FIN),$$

for $\alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 1\}$.

On the other hand, if one can use any RE language as the control program, then both classes of SC P systems have the same computational power. In particular, from Theorem 4 and Corollary 3, one gets the following result.

Theorem 12

$$(pro)N^{(1)}P_m(\alpha, RE) = N^{(1)}P_m(\alpha, RE) = NRE,$$

for $\alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 1\}$.

7 Concluding Remarks and Open Problems

We have introduced and investigated SC P systems where the computations are driven by control strings (present in their skin region at the beginning of computations). We have mainly investigated fully-promoted SC P systems, where all the rules are promoted (hence controlled by the control strings). Most of the results proved in this paper concern systems operating in mode (1), although this is just a matter of convenience, because we have proved the equivalence between both operating modes (under some conditions).

Table 1 gives an overview of the results obtained for fully-promoted SC P systems operating in mode (1) and collecting the result in the # way.

Table 1. Computational power of fully-promoted SC P systems operating in mode (1) and collecting the result in the # way. Rows specify the types of evolution rules, and the columns specify the types of control programs.

	RE	**CF**	**REG**	**FIN**
$ncoo$	NRE	$NE(cfc)T0L$	$NET0L$	$NFIN$
$cat_i, i \geq 1$	NRE	NRE	NRE	$NFIN$
coo	NRE	NRE	NRE	$NFIN$

The results obtained for fully-promoted SC P systems operating in mode (1) and collecting the result in the standard way are summarized in Table 2.

Table 2. Computational power for fully-promoted SC P systems operating in mode (1) and collecting the result in the standard way. Again, rows specify the types of evolution rules, and the columns specify the types of control programs.

	RE	**CF**	**REG**	**FIN**
$ncoo$	NRE	$N(cfc)T0L$	$N(rc)T0L$	$NFIN$
$cat_i, i \geq 1$	NRE	$\supseteq N(cfc)T0L$	$\supseteq N(rc)T0L$	$NFIN$
coo	NRE	$\supseteq N(cfc)T0L$	$\supseteq N(rc)T0L$	$NFIN$

Several problems, mainly concerning non fully-promoted systems, remain open. Are non-fully promoted SC P systems more powerful than fully-promoted

SC P systems? The answer is positive for SC P systems operating in mode (1) and having a finite control program (Theorem 11). We conjecture that the strict inclusion also holds when the control program is regular and the result is collected in the standard way.

Another open problem is to find a non-trivial upper bound for the generative power of fully-promoted SC P systems operating in mode (1), collecting the result in the standard way, and using cooperative or catalytic rules (see Table 2). We only know that these classes of systems can generate at least the family of length sets of languages from $(rc)T0L$ (if the control program is regular) and from $(cfc)T0L$ (if the control program is context-free). We doubt that these two classes are universal – as a matter of fact they may be incomparable with the classical Chomsky classes.

Finally, another interesting issue to be investigated is having the control programs produced by another bio-inspired generative device (as for instance, another membrane system, or a DNA-based system).

Acknowledgments

The authors are indebted to the European Research Network SegraVis for supporting this research. R.B. is supported by the Netherlands Organization for Scientific Research (NWO) project 635.100.006 "VIEWS".

References

1. P. Bottoni, C. Martín-Vide, Gh. Păun, G. Rozenberg: Membrane Systems with Promoters/Inhibitors. *Acta Informatica*, 38, 10 (2002), 695–720.
2. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin, 1989.
3. R. Freund, L. Kari, M. Oswald, P. Sosik: Computationally Universal P Systems without Priorities: Two Catalysts are Sufficient. *Theoretical Computer Science*, 330, 2 (2005), 251–266.
4. S. Ginsburg, G. Rozenberg: T0L Schemes and Control Sets. *Information and Control*, 27 (1974), 109–125.
5. M. Ionescu, D. Sburlan: On P Systems with Promoters/Inhibitors. *Journal of Universal Computer Science*, 10, 5 (2004), 581–599.
6. Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
7. Gh. Păun, G. Rozenberg: A Guide to Membrane Computing. *Theoretical Computer Science*, 287, 1 (2002), 73–100.
8. G. Rozenberg, A. Salomaa: *The Mathematical Theory of L Systems*. Academic Press, Inc. Orlando, FL, USA, 1980.
9. G. Rozenberg, A. Salomaa (eds.): *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.
10. A. Salomaa: *Formal Languages*. Academic Press, New York, 1973.
11. P systems web page: `http://psystems.disco.unimib.it/`

A Case Study in (Mem)Brane Computation: Generating Squares of Natural Numbers

Nadia Busi[1] and Miguel A. Gutiérrez-Naranjo[2]

[1] Dipartimento di Scienze dell'Informazione - Università di Bologna
Mura Anteo Zamboni 7, I-40127 Bologna, Italy
busi@cs.unibo.it
[2] Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
magutier@us.es

Abstract. The aim of this paper is to start an investigation and a comparison of the expressiveness of the two most relevant formalisms inspired by membranes interactions, namely, P systems and Brane Calculi. We compare the two formalisms with respect to their ability to act as generator devices. In particular, we show different ways of generating the set $\mathcal{L} = \{n^2 \mid n \geq 1\}$ in P systems and in Brane Calculi.

1 Introduction

Natural Computing studies new computational paradigms inspired from various well known natural phenomena in physics, chemistry, and biology. It abstracts the way in which nature computes, conceiving new computing models. There are several fields in Natural Computing that are now well established. Among them, we mention *Genetic algorithms* introduced by J. Holland [7] that is inspired by natural evolution and selection in order to find a good solution in a large set of feasible candidate solutions, *Neural Networks* introduced by W.S. McCulloch and W. Pitts [8] which is based on the interconnections of neurons in the brain, and *DNA-based* molecular computing, that was born when L. Adleman [1] published a solution to an instance of the Hamiltonian path problem by manipulating DNA strands in a lab.

This paper is devoted to a new field in Natural Computing. Starting from the structure and functioning of cells as living organisms able to process and generate information, two different branches of Natural Computing were recently initiated: *Membrane Computing* and *Brane Calculi*.

Membrane Computing was introduced by Gh. Păun in [9]; a comprehensive presentation[1] can be found at [11]. The devices of this model are called *P systems*. Roughly speaking, a P system consists of a membrane structure, in the compartments of which one places multisets of objects which evolve according to given rules in a synchronous non-deterministic maximally parallel manner.

[1] A layman-oriented introduction can be found in [10] and further bibliography at [14].

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 233–249, 2006.

Brane Calculi were introduced by L. Cardelli in [4] on the assumption that in living cells membranes are not merely containers, they are highly dynamic and participate actively in the cell life. In this way, computation happens on the membrane, not inside of it.

The first attempt of bridging the two research areas was made in [6] by the *fathers* of the two disciplines, L. Cardelli and Gh. Păun. As they point out, Membrane Computing and Brane Calculi *have different objectives and develop in different directions. While Membrane Computing tries to abstract computing models, in the Turing sense, from the structure and the functioning of the cell (...), Brane Calculi pay more attention to the fidelity to the biological reality (...).*

In that paper [6], four basic operations from Brane Calculi, namely, *pino*, *exo*, *mate* and *drip*, are expressed in terms on the Membrane Computing formalism and the Turing completeness of systems which use the *mate*, *drip* operations is shown. The Turing universality of Brane Calculi (in fact, by using only *phago* and *exo* operations) was proved in [3]. Recently, it has been proved that P systems with *mate* and *drip* operations and using at most five membranes during any step of a computation are universal (see [2]). This result improves a similar one from [6] were eleven membranes are used.

In some sense, in this paper we cross the bridge in the other way. Instead of expressing Brane Calculi operations in terms of the Membrane Computing formalism, we take a problem from computability, the generation of a set of numbers, and we show how it can be handled both in Membrane Computing and in Brane Calculi.

The paper is organized as follows: first the case study, i.e., the set $\mathcal{L} = \{n^2 \mid n \geq 1\}$ and some considerations with respect to the codifications are fixed in the next section. In Section 3, two different Membrane Computing devices that generate $\mathcal{L}$ are shown. Inspired on the second Membrane Computing design, two Brane Calculi devices that generate $\mathcal{L}$ are presented in Section 4. Some final remarks are presented in the last section.

2 The Case Study

Computational devices can be designed in order to perform different tasks. Among such tasks, they can be designed to solve decision problems (I_X, θ_X) where I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total boolean function over I_X. In a more general case, the function is not boolean and the problem consists on the computation of a function f from I_X onto a general set S.

Another type of tasks is the generation of various sets (of numbers, vectors, strings, etc.). Due to the nondeterminism, several different computations are obtained and some piece of information is considered as the *output*. Collecting all (acceptable) outputs, we get a set (of numbers, vectors, strings, etc.) generated by the computation device.

In order to fix ideas, let us consider the case study used in this paper. We will consider the set $\{n^2 \mid n \geq 1\}$. For its generation, we will design appropriate devices in the computational models *Membrane Computing* and *Brane Calculi*. Such devices are non-deterministic and several computations can be performed from the starting point. In each device, a piece of information will be considered the *output* of the system. In the case of Membrane Computing, the *output* is codified as the number of objects inside a fixed membrane in a halting configuration. In the Brane Calculi device, the *output* is codified as the number of membranes of a specific kind that are present in the system in a halting computation. The set of all possible outputs of the device is exactly $\mathcal{L} = \{n^2 \mid n \geq 1\}$. In this way, $\mathcal{L}$ is the set generated by the device.

3 Membrane Computing

In Membrane Computing, many different types of rules and different semantics have been presented. The choice of these rules and semantics lead us to different models of P systems. In this section we present two P systems constructed in two different models that generate the set $\{n^2 \mid n \geq 1\}$.

In these examples several types of rules are used (O is the alphabet of *objects*, H is a finite set of *labels*, and λ is the empty string):

- *Object evolution rules* $[a \to v]_h$ where $h \in H$, $a \in O$, and v is a string over O describing a multiset of objects. They are associated with membranes and depending only on the label of the membrane. Using such a rule means that an object a evolves to the multiset v inside the membrane with label h.
- *Cooperation rules:* $[v \to w]_h$ where $h \in H$ and v, w are string over O describing a multisets of objects. This rule is similar to the previous one, but in this type, the rule is triggered by a multiset of objects whereas in an *object evolution rule* only one object is necessary for triggering it.
- *Dissolution rules:* $[a]_h \to b$ where $h \in H$, $a \in O$, $b \in O \cup \{\lambda\}$. The object a inside the membrane labeled with h produces the dissolution of the membrane and it is transformed into the object b. This object b together with the remaining objects in the membrane h are placed inside the surrounding membrane.
- *Send-in communication rules:* $a[\ ]_h \to [b]_h$ where $h \in H$, $a, b \in O$. An object a out of the membrane labeled with h is sent into the membrane and transformed into b.
- *Send-out communication rules:* $[a]_h \to [\ ]_h\, b$ where $h \in H$, $a, b \in O$. This is dual to the previous case. An object a inside the membrane labeled with h is sent out of the membrane and transformed into b.

Rules are applied according to the following principles:

- Rules are used as usual in the framework of Membrane Computing, that is, in a maximally parallel way. In one step, each object in a membrane can only be used for one rule (non-deterministically chosen when there are

several possibilities), but any object which can evolve by a rule of any form must do it (with the restrictions indicated below).
- If a membrane is dissolved, its content (multiset and internal membranes) becomes part of the immediately external one. The skin membrane is never dissolved.
- All the elements which are not involved in any of the operations to be applied remain unchanged.
- Several rules can be applied to different objects in the same membrane simultaneously. The exception are the division rules since a membrane can be dissolved only once.

In order to generate a set, an output membrane is fixed and the number of objects in it is counted when the system halts. The number of objects can vary from one computation to other due to the nondeterminism of the system. In the next examples, the set of numbers obtained in the output membrane, i.e., the generated set, is $\{n^2 \mid n \geq 1\}$.

3.1 Cooperation and Priorities

The first P system that we show is taken from [11] (p. 75) and it uses two of the most powerful features in P systems. The first one is the use of rules with *cooperation* between objects as described above. This type of rules are not triggered by the occurrence of only one object, but two or more objects are necessary in order to trigger the rule. The second feature is the priority among rules. In the general framework of Membrane Computing, if two rules can be applied, one of them is chosen in a non-deterministic way. If a priority between rules is added, we decrease the non-determinism, since we have a precedence between them.

With the notation fixed above, the P system is $\Pi = (O, H, \mu, w_1, w_2, w_3, 1, R)$ where $O = \{a, b, d, e, f\}$ is the set of objects, $H = \{1, 2, 3\}$ is the set of labels, $\mu = [\,[\,[\,]_3\,]_2\,]_1$ is the membrane structure, $w_1 = \emptyset$, $w_2 = \emptyset$, $w_3 = af$ are the multisets placed in the membranes at the starting point, 1 is the label of the *output membrane*, and R is the set of rules:

Rule 1: $[a \to ab]_3$ **Rule 5:** $[b \to d]_2$
Rule 2: $[a]_3 \to b$ **Rule 6:** $[d \to de]_2$
Rule 3: $[f \to ff]_3$ **Rule 7:** $[ff \to f]_2$
Rule 4: $[d]_1 \to d\,[\,]_1$ **Rule 8:** $[f]_2 \to \lambda$

with the priority

$$(\textbf{Rule 7: } [ff \to f]_2) > (\textbf{Rule 8: } [f]_2 \to \lambda)$$

Rules **1**, **3**, **5** and **6** are *object evolution rules*. Rule **7** is a cooperation rule: we need *two* objects f in order to trigger the rule. Rules **2** and **8** are *dissolution rules*. Finally, rule **4** is a *send-out communication rule*.

The computation is performed as follows. In the initial configuration we only have objects af in the membrane labeled with 3.

$$C_0 = [\,[\,[\,af\,]_3\,]_2\,]_1$$

Due to rule **3**, the object f deterministically evolves to ff. For the object a we have two possibilities: By application of rule **1**, the object a evolves to ab or by applying rule **2**, membrane 3 dissolves. If we iterate the use of rules **1** and **3**, after n steps, $n \geq 0$, we get n occurrences of b, one copy of a, and 2^n occurrences of f in membrane 3.

$$\begin{aligned} C_1 &= [\,[\,[\,abf^2\,]_3\,]_2\,]_1 \\ C_2 &= [\,[\,[\,ab^2f^4\,]_3\,]_2\,]_1 \\ &\dots \\ C_n &= [\,[\,[\,ab^nf^{2^n}\,]_3\,]_2\,]_1 \end{aligned}$$

If then rule **2** is chosen, the membrane labeled with 3 is dissolved *after* the evolution of f. With the dissolution, the 2^{n+1} copies of object f and the $n+1$ copies of b become occurrences of objects of membrane 2.

$$C_{n+1} = [\,[\,b^{n+1}f^{2^{n+1}}\,]_2\,]_1$$

In one step, the $n+1$ copies of b are transformed into $n+1$ copies of d by rule **5**, while the number of occurrences of f is halved.

$$C_{n+2} = [\,[\,d^{n+1}f^{2^n}\,]_2\,]_1$$

In the next step each occurrence of d introduces one occurrence of e and the number of occurrences of f is halved again.

$$C_{n+3} = [\,[\,d^{n+1}e^{n+1}f^{2^{n-1}}\,]_2\,]_1$$

After n applications of rule **7**, $[ff \to f]_2$, only one copy of object f is present in membrane labeled with 2. In the meantime, rule **6** is applied $n+1$ times in each step.

$$\begin{aligned} C_{n+4} &= [\,[\,d^{n+1}e^{2(n+1)}f^{2^n-2}\,]_2\,]_1 \\ C_{n+5} &= [\,[\,d^{n+1}e^{3(n+1)}f^{2^n-3}\,]_2\,]_1 \\ &\dots \end{aligned}$$

Following the priority relation, rule **7** $[ff \to f]_2$ is used as much as possible; when only one object f remains, rule **8** is used.

$$\begin{aligned} C_{2n+2} &= [\,[\,d^{n+1}e^{n(n+1)}f\,]_2\,]_1 \\ C_{2n+3} &= [\,d^{n+1}e^{(n+1)^2}\,]_1 \end{aligned}$$

With the dissolution of membrane 2, all the objects d become objects of membrane 1. In the next step, rule **4** is applied $n+1$ times and all copies of d are sent out to the environment.

$$C_{2n+4} = [\,e^{(n+1)^2}\,]_1\,d^{n+1}$$

No further step is possible and the computation stops. In the membrane labeled with 1 we have $(n+1)(n+1)$ copies of object e for some $n \geq 0$, hence the set generated is $\{n^2 \mid n \geq 1\}$.

3.2 A Simplified Solution

Now we present a *new* solution to the same problem. We do not use cooperation or priorities. Only send-in communication, dissolution and object evolution rules are applied. The design is based on the well-known property of natural numbers

$$\sum_{k=0}^{n}(2k+1) = (n+1)^2 \quad \text{for all } n \geq 0$$

The P system is the following: $\Pi = (O, H, \mu, w_e, w_r, w_s, r, R)$ with the set of objects $O = \{a, b, c, z\}$, the set of labels $H = \{e, r, s\}$, the membrane structure $\mu = [\,[\,]_e\,[\,]_r\,]_s$. The initial multisets are $w_e = a^2bz$, $w_r = \emptyset$ and $w_s = \emptyset$, i.e., the membranes s and r are empty and there exist two copies of a and one copy of b and z in the membrane e. The *output membrane* is labeled with r and the set of rules R is the following:

Rule 1: $[\,a \to ab\,]_e$ **Rule 5:** $[\,a \to \lambda\,]_s$
Rule 2: $[\,b \to bc\,]_e$ **Rule 6:** $[\,b \to \lambda\,]_s$
Rule 3: $[\,z \to z\,]_e$ **Rule 7:** $c\,[\,]_r \to [\,c\,]_r$
Rule 4: $[\,z\,]_e \to \lambda$

Note that the only non-determinism in this example is produced by the object z. This object can trigger two rules. The first one is $[\,z \to z\,]_e$ which represents that the object z inside the membrane e does not change. The second one is $[\,z\,]_e \to \lambda$ which means that the object z dissolves the membrane e. The collateral effect of the application of this rule is that the remaining objects in e are sent to s.

The initial configuration is $C_0 = [\,[\,a^2bz\,]_e\,[\,]_r\,]_s$. In the first step the two objects a evolve according to the rule **1**, $[\,a \to ab\,]_e$, and the object b evolves following the rule **2**, $[\,b \to bc\,]_e$. These evolutions are deterministic. For the object z we have two options, rules **3** and **4**. Let us suppose that z remains unchanged following rule **3**, $[\,z \to z\,]_e$. We obtain the configuration $C_1 = [\,[\,a^2b^3cz\,]_e\,[\,]_r\,]_s$. Let us suppose that in the next steps the object z does not dissolve the membrane e. We obtain $C_2 = [\,[\,a^2b^5c^4z\,]_e\,[\,]_r\,]_s$, $C_3 = [\,[\,a^2b^7c^9z\,]_e\,[\,]_r\,]_s$,... and in general, if the element z does not dissolves the membrane e, in the n-th ($n \geq 1$) step we reach the configuration

$$C_n = [\,[\,a^2b^{2n+1}c^{n^2}z\,]_e\,[\,]_r\,]_s$$

Let us now suppose that in the n-th step the object z dissolves the membrane e by using rule **4**. Since the dissolution is considered *after* the evolution of objects a and b, we reach the configuration

$$C_{n+1} = [\,a^2b^{2(n+1)+1}c^{(n+1)^2}z\,[\,]_r\,]_s \quad n \geq 0$$

One of the effects of the dissolution is that the objects a, b, and c are now in the membrane s. On one hand the rules $[\, a \to \lambda \,]_s$ and $[\, b \to \lambda \,]_s$ are triggered in the next step, so objects a and b disappear. On the other hand, objects c are in the region surrounding the membrane r, so the communication rule $c\,[\,]_r \to [\, c\,]_r$ are applied and all the elements c go into membrane r. In this way, the next configuration is $C_{n+2} = [\,[c^{(n+1)^2}\,]_r\,]_s$ with $n \geq 0$.

No more rules can be applied, so this is a halting configuration and we have computed the number n^2 with $n \geq 1$ (encoded by the elements c) in the output membrane.

4 Brane Calculi

In this section we tackle the problem of generating the set $\{n^2 \mid n \geq 1\}$ in Brane Calculi.

Brane Calculi [4] are a family of process calculi proposed for modeling the behavior of biological membranes. In a process algebraic setting, Brane Calculi represent an evolution of BioAmbients [12], a variant of Mobile Ambients [5] based on a set of biologically inspired primitives of interaction. The main novelty of Brane calculi consists in the fact that the active entities reside on membranes, and not inside membranes.

In this paper we are interested in the membrane operations of two basic instances of Brane calculi proposed in [4]: the Phago/Exo/Pino (PEP) and the Mate/Bud/Drip (MBD) calculi.

The interaction primitives of PEP are inspired by *endocytosis* (the process of incorporating external material into a cell by engulfing it with the cell membrane) and *exocytosis* (the reverse process). A relevant feature of such primitives is *bitonality*, a property ensuring that there will never be a mixing of what is inside a membrane with what is outside, although external entities can be brought inside if safely wrapped by another membrane. As endocytosis can engulf an arbitrary number of membranes, it turns out to be a rather uncontrollable process. Hence, it is replaced by two simpler operations: *phagocytosis*, that is engulfing of just one external membrane, and *pinocytosis*, that is engulfing zero external membranes.

The primitives of MBD are inspired by membrane fusion (mate) and fission (mito). Because membrane fission is an uncontrollable process that can split a membrane at an arbitrary place, it is replaced by two simpler operations: *budding*, that is splitting off one internal membrane, and *dripping*, that consists in splitting off zero internal membranes. An encoding of the MBD primitives in PEP is provided in [4].

4.1 Basic Brane Calculi: Syntax and Semantics

In this section we recall the syntax and the semantics of Brane Calculi [4]. A system consists of nested membranes, and a process is associated to each membrane.

Definition 1. *The set of systems is defined by the following grammar:*

$$P, Q \quad ::= \quad \diamond \mid P \circ Q \mid !P \mid \sigma(\!| P |\!)$$

The set of membrane processes is defined by the following grammar:

$$\sigma, \tau \quad ::= \quad 0 \mid \sigma|\tau \mid !\sigma \mid a.\sigma$$

Variables a, b range over actions that will be detailed later.

The term $\diamond$ represents the empty system; the parallel composition operator on systems is $\circ$. The replication operator ! denotes the parallel composition of an unbounded number of instances of a system. The term $\sigma(\!| P |\!)$ denotes the membrane that performs process σ and contains system P.

The term 0 denotes the empty process, whereas | is the parallel composition of processes; with $!\sigma$ we denote the parallel composition of an unbounded number of instances of process σ. Term $a.\sigma$ is a guarded process: after performing the action a, the process behaves as σ.

We adopt the following abbreviations: with a we denote $a.0$, with $(\!| P |\!)$ we denote $0(\!| P |\!)$, and with $\sigma(\!|\ |\!)$ we denote $\sigma(\!| \diamond |\!)$.

The structural congruence relation on systems and processes is defined as follows:[2]

Definition 2. *The structural congruence $\equiv$ is the least congruence relation satisfying the following axioms:*

$$\begin{array}{ll}
P \circ Q \equiv Q \circ P & \sigma \mid \tau \equiv \tau \mid \sigma \\
P \circ (Q \circ R) \equiv (P \circ Q) \circ R & \sigma \mid (\tau \mid \rho) \equiv (\sigma \mid \tau) \mid \rho \\
P \circ \diamond \equiv P & \sigma \mid 0 \equiv \sigma \\
 & \\
!\diamond \equiv \diamond & !0 \equiv 0 \\
!(P \circ Q) \equiv !P \circ !Q & !(\sigma \mid \tau) \equiv !\sigma \mid !\tau \\
!!P \equiv !P & !!\sigma \equiv !\sigma \\
P \circ !P \equiv !P & \sigma \mid !\sigma \equiv !\sigma \\
 & \\
0(\!| \diamond |\!) \equiv \diamond &
\end{array}$$

Definition 3. *The basic reaction rules are the following:*

$$(\texttt{par}) \quad \frac{P \ \rightarrow \ Q}{P \circ R \ \rightarrow \ Q \circ R} \qquad (\texttt{brane}) \quad \frac{P \ \rightarrow \ Q}{\sigma(\!| P |\!) \ \rightarrow \ \sigma(\!| Q |\!)}$$

$$(\texttt{strucong}) \quad \frac{P' \equiv P \qquad P \ \rightarrow \ Q \qquad Q \equiv Q'}{P' \ \rightarrow \ Q'}$$

[2] With abuse of notation we use $\equiv$ to denote both structural congruence on systems and structural congruence on processes.

Rules (par) and (brane) are the contextual rules that permit to a system to execute also if it is in parallel with another process or if it is inside a membrane, respectively. Rule (strucong) ensures that two structurally congruent systems have the same reactions.

With $\rightarrow^*$ we denote the reflexive and transitive closure of a relation $\rightarrow$.

We say that a system P is *deterministic* iff for all P', P'': if $P \rightarrow P'$ and $P \rightarrow P''$ then $P' \equiv P''$. We say that P has a *halting computation* (or a deadlock) if there exists Q such that $P \rightarrow^* Q$ and $Q \not\rightarrow$.

The system P' is a *derivative* of the system P if $P \rightarrow^* P'$; the set of *derivatives* of a system P is denoted by $Deriv(P)$.

The Phago/Exo/Pino Calculus (PEP). The PEP calculus is inspired by endocytosis/exocytosis. Endocytosis is the process of incorporating external material into a cell by "engulfing" it with the cell membrane, while exocytosis is the reverse process. As endocytosis can engulf an arbitrary amount of material, giving rise to an uncontrollable process, in [4] two more basic operations are used: *phagocytosis*, engulfing just one external membrane, and *pinocytosis*, engulfing zero external membranes.

Definition 4. *Let Name be a denumerable set of ambient names, ranged over by $n, m, \ldots$. The set of actions of PEP is defined by the following grammar:*

$$a \quad ::= \quad \lhd_n \mid \lhd_n^\perp(\sigma) \mid \rhd_n \mid \rhd_n^\perp \mid \circledcirc(\sigma)$$

Action $\lhd_n$ denotes phagocytosis; the co-action $\rhd_n^\perp$ is meant to synchronize with $\lhd_n$; names n are used to pair-up related actions and co-actions. The co-phago action is equipped with a process σ, this process will be associated to the new membrane that engulfs the external membrane. Action $\rhd_n$ denotes exocytosis, and synchronizes with the co-action $\rhd_n^\perp$. Exocytosis causes an irreversible mixing of membranes. Action $\circledcirc$ denotes pinocytosis. The pino action is equipped with a process σ: this process will be associated to the new membrane, that is created inside the membrane performing the pino action.

Definition 5. *The reaction relation for PEP is the least relation containing the following axioms, and satisfying the rules in Definition 3:*

(phago) $\quad \lhd_n.\sigma|\sigma_0(\!|\, P \,|\!) \;\circ\; \rhd_n^\perp(\rho).\tau|\tau_0(\!|\, Q \,|\!) \rightarrow \tau|\tau_0(\!|\, \rho(\!|\, \sigma|\sigma_0(\!|\, P \,|\!) \,|\!) \circ Q \,|\!)$

(exo) $\quad \rhd_n^\perp.\tau|\tau_0(\!|\, \lhd_n.\sigma|\sigma_0(\!|\, P \,|\!) \circ Q \,|\!) \rightarrow P \;\circ\; \sigma|\sigma_0|\tau|\tau_0(\!|\, Q \,|\!)$

(pino) $\quad \circledcirc(\rho).\sigma|\sigma_0(\!|\, P \,|\!) \rightarrow \sigma|\sigma_0(\!|\, \rho(\!|\, \,|\!) \circ P \,|\!)$

The Mate/Bud/Drip Calculus (MBD). The MBD calculus is inspired by membrane fusion and splitting. To make membrane splitting more controllable, in [4] two more basic operations are used: *budding*, consisting in splitting off one internal membrane, and *dripping*, consisting in splitting off zero internal membranes. Membrane fusion, or merging, is called *mating*.

Definition 6. *The set of actions of MBD is defined by the following grammar:*

$$a \quad ::= \quad mate_n \mid mate_n^{\perp} \mid bud_n \mid bud_n^{\perp}(\sigma) \mid drip(\sigma)$$

Actions $mate_n$ and $mate_n^{\perp}$ will synchronize to obtain membrane fusion. Action bud_n permits to split one internal membrane, and synchronizes with the co-action $bud_n^{\perp}$. Action *drip* permits to split off zero internal membranes. Actions $bud^{\perp}$ and *drip* are equipped with a process σ, that will be associated to the new membrane created by the membrane performing the action.

Definition 7. *The reaction relation for MBD is the least relation containing the following axioms, and satisfying the rules in Definition 3:*

(mate) $\quad mate_n.\sigma|\sigma_0\langle\!|\, P \,|\!\rangle \circ mate_n^{\perp}.\tau|\tau_0\langle\!|\, Q \,|\!\rangle \rightarrow \sigma|\sigma_0|\tau|\tau_0\langle\!|\, P \circ Q \,|\!\rangle$

(bud) $\quad bud_n^{\perp}(\rho).\tau|\tau_0\langle\!|\, bud_n.\sigma|\sigma_0\langle\!|\, P \,|\!\rangle \circ Q \,|\!\rangle \rightarrow \rho\langle\!|\, \sigma|\sigma_0\langle\!|\, P \,|\!\rangle\!|\rangle \circ \tau|\tau_0\langle\!|\, Q \,|\!\rangle$

(drip) $\quad drip(\rho).\sigma|\sigma_0\langle\!|\, P \,|\!\rangle \rightarrow \rho\langle\!|\,|\!\rangle \circ \sigma|\sigma_0\langle\!|\, P \,|\!\rangle$

In [4] it is shown that the operations of mating, budding and dripping can be encoded in PEP.

For the sake of simplicity, in the present paper we consider a basic calculus containing the membrane interaction primitives of both the PEP and the MBD calculi. As the primitives of MBD can be encoded in PEP, we conjecture that the system described in the following part of the paper can be encoded in an equivalent system that makes use of the PEP primitives only.

4.2 Computing $\{n^2 \mid n \geq 1\}$ in Brane Calculi

Now we show how to model our case study in Brane Calculi. Our solution is inspired by the simplified solution in Subsection 3.2. When moving from P systems to Brane Calculi, two main problems arise.

The first problem is concerned with the fact that in Basic Brane Calculi there are no objects/proteins floating inside the membranes. Hence, we need an alternative representation of the output of our system. In the solution based on P systems presented in Subsection 3.2, the natural number n is represented as n occurrences of object c inside membrane r. Here the idea is to represent the output as a family of membranes with a particular process C on them, such that process C can be distinguished by other processes residing on other auxiliary membranes.

A second major problem is concerned with the interleaving semantics of Brane Calculi. We note that the maximal parallelism semantics of P systems is a very powerful synchronization mechanism. This ensures that – at each computational step – for each occurrence of object b a new object c is created and for each occurrence of object a a new object b is created. If we simply encode each object a (resp. b, c) with a membrane $A\langle\!|\,|\!\rangle$ (resp. $B\langle\!|\,|\!\rangle$, $C\langle\!|\,|\!\rangle$), thus obtaining a flat multiset of membranes, then for mimicking a computational step of the corresponding P

system we need to perform a synchronization among an unbounded number of membranes, and this seems to be a very difficult task in Brane Calculi. On the other hand, it is quite easy to synchronize an a priori fixed number of membranes. To solve this problem, we decided to move from the flat structure of membranes proposed above (and consisting in a multiset of membranes $A(\!|\ |\!)$, $B(\!|\ |\!)$, and $C(\!|\ |\!)$ contained in the same surrounding membrane) to a hierarchical structure.

We start presenting a simplified version of the solution, where the output of the system is represented by the number of occurrences of C appearing in the whole structure of the system, and not inside a specific membrane. Then, we present a more elegant solution where the output of the system is represented by the number of occurrences of C contained in a specific membrane.

Solution with output scattered in the whole system. The initial system consists of an external membrane, containing two instances of membranes representing an encoding of object a and one brane representing an encoding of object b, as depicted in Figure 1 (the need for the auxiliary membranes decorated with processes X, Ta and Tb will be clarified in the following).

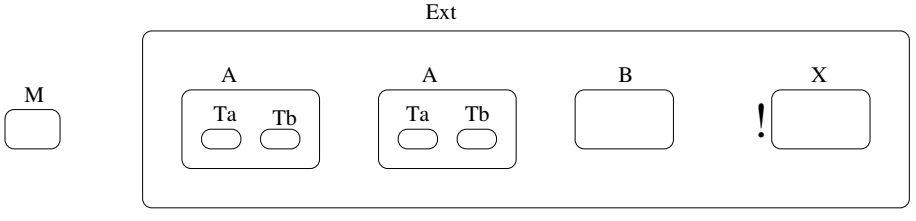


Fig. 1. The initial membrane system (with $M = mate_n^{\perp}$)

We mimic a single maximal parallelism computational step of the P system in Subsection 3.2 by the following sequence of steps: each membrane encoding object b creates – by dripping – a new membrane representing an encoding of c; each membrane encoding object a is surrounded by a newly created membrane representing a and containing a new instance of a membrane representing b.

An evolution of the representation of an object a as a nested family of membranes is reported in Figure 2.

The representations of objects a and b are arranged in a hierarchical structure: there exists a membrane with process A (and representing object a) surrounding both a membrane with process B (representing object b) and another membrane with process A' (surrounded by another membrane with process E – such a membrane is created during the phagocytosis to preserve bitonality and cannot be avoided). The membrane with object A' contains a membrane decorated with B and another membrane E containing a membrane A', and so on. The most internal instance of membrane decorated with A' contains the two terminal membranes Ta and Tb.

A maximal parallelism computational step of the P system in Subsection 3.2 is mimicked in the following way: the external membrane – with process Ext –

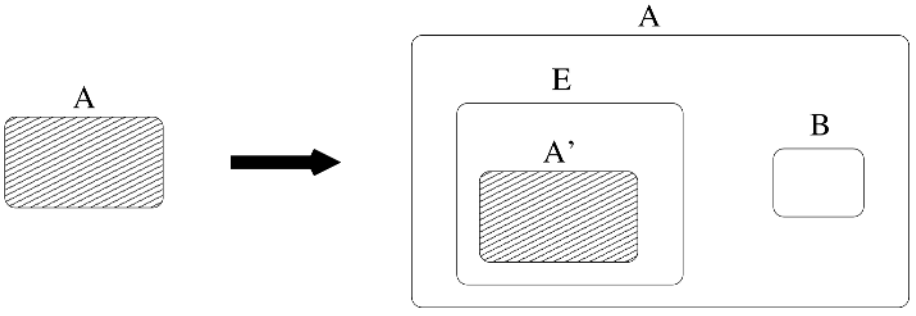


Fig. 2. The evolution of the system encoding object a

sends one (asynchronous) signal to each of its children. The child membrane with process B reacts to the signal by spawning a new child membrane with process C, and sends a signal to the external brane to communicate that it has finished its task. Each child membrane with process A reacts in the following way:

- first of all, the A membrane sends two signals to its children – decorated with B and E – that will be used to wake up the instances of membranes decorated with B inside the hierarchical structure (each of such B membranes will spawn a new C membrane);
- then it waits for two signals from its children, to acknowledge the end of the creation of new copies of C by the B membranes in the hierarchical structure;
- now, a new membrane is created, and the A membrane enters this new membrane by phagocytosis and spawns a new membrane with process B;
- finally, the A membrane sends a signal to the external membrane to acknowledge the end of its task, and evolves to a membrane with process A'.

Before presenting the definition of the system, we show how to obtain asynchronous communication between a father and a child membrane. If the father membrane wants to send a signal to one of its children, it produces by pinocytosis a bubble with process $mate_x^{\perp}$; the child accepts this signal by performing an action $mate_x$. On the other hand, if a child wants to send a signal to its father, it produces by dripping a bubble with process $\circlearrowleft_x$; the father receives this signal by performing an action $\circlearrowleft_x^{\perp}$.

Formally, the system is defined as follows:

$$\begin{aligned} mate_n^{\perp}(\!|\ \ |\!) \circ Ext(\!| & A(\!| Ta(\!|\ \ |\!) \circ Tb(\!|\ \ |\!) \,|\!) \circ \\ & A(\!| Ta(\!|\ \ |\!) \circ Tb(\!|\ \ |\!) \,|\!) \circ \\ & B(\!|\ \ |\!) \circ \\ & !(X(\!|\ \ |\!)) \,|\!) \end{aligned}$$

So, we have a big membrane containing two copies of A and one copy of B, plus the membrane $mate_n^{\perp}(\!|\ \ |\!)$. The membrane $mate_n^{\perp}(\!|\ \ |\!)$ is a trigger that fuses with the big membrane: if the fusion is performed by the first $mate_n$ action of

Ext, then some new copies of C are produced; otherwise, the system ends. As we already said before, the output of the system is represented by the number of occurrences of C appearing in the whole structure of the system, and not inside a specific membrane.

The process Ext is the following:

$$\begin{aligned} Ext = {} & !mate_n.\ \circledcirc(mate^{\perp}_{a_s}).\ \circledcirc(mate^{\perp}_{a_s}).\ \circledcirc(mate^{\perp}_{b_s}).\circlearrowleft^{\perp}_{a_f}.\circlearrowleft^{\perp}_{a_f}. \\ & \circlearrowleft^{\perp}_{b_f}.drip(mate^{\perp}_{n}) \mid \\ & mate_n.0 \end{aligned}$$

The program Ext triggers the two copies of A and B by producing three bubbles by pinocytosis that can fuse with the two instances of A and with B. The membrane B simply produces a child bubble labeled with C then signals the termination of this task to the external membrane. In this simplified version of the solution, C may be any process that can be distinguished from the others.

The evolution of membrane A is depicted in Figure 2; here we give a more detailed description of the behavior of such a kind of membrane.

First of all, the membrane A sends a signal to its children: at the beginning, this membrane has two dummy children (represented by systems Ta and Tb) that simply send back the signal; however, during the computation the last created membrane A has to send a signal to its children to permit to its descendants of kind B to produce new copies of C. Thus, membrane A sends a signal with label a_s to its child with process E and a signal with label b_s to its child with process B to trigger the starting of the execution of a computational step by the two children. Then, the membrane A waits for two signals: a signal with label a_f from its child E (meaning that all the B descendants have spawn a new copy of C) and a signal with label b_f from its child B (meaning that B has spawn a new copy of C). After the membrane A has received these two signals from its children, membrane A creates a new sibling bubble decorated with D, then A enters the D bubble (note that phagocytosis creates a new membrane surrounding A inside D; this causes the necessity to propagate signals across this membrane, that has process E). After A enters D, D creates a child with process B by pinocytosis, and then signals that it has finished its task to its father, and then, by fusing with a copy of an X membrane, it becomes a membrane with program A.

The definitions of the remaining systems and processes are as follows:

$$\begin{aligned} A &= mate_{a_s}.\ \circledcirc(mate^{\perp}_{a_s}).\ \circledcirc(mate^{\perp}_{b_s}).\circlearrowleft^{\perp}_{a_f}.\circlearrowleft^{\perp}_{b_f}.drip(D).\circlearrowleft_{d}.A' \\ A' &= !mate_{a_s}.\ \circledcirc(mate^{\perp}_{a_s}).\ \circledcirc(mate^{\perp}_{b_s}).\circlearrowleft^{\perp}_{a_f}.\circlearrowleft^{\perp}_{b_f}.drip(\circlearrowleft_{a_f}) \\ D &= \circlearrowleft^{\perp}_{d}(E).\ \circledcirc(B).drip(\circlearrowleft_{a_f}).mate^{\perp}_{x} \\ X &= mate_x.A \\ E &= !mate_{a_s}.\ \circledcirc(mate^{\perp}_{a_s}).\circlearrowleft^{\perp}_{a_f}.drip(\circlearrowleft_{a_f}) \\ B &= !mate_{b_s}.\ \circledcirc(C).drip(\circlearrowleft_{b_f}) \\ Ta &= (!mate_{a_s}.drip(\circlearrowleft_{a_f})) \\ Tb &= (!mate_{b_s}.drip(\circlearrowleft_{b_f})) \end{aligned}$$

Solution with output contained in a specific membrane. Now we show how to put the encoding of the output of the system inside a single membrane, with process Res. First of all, we surround the system by two membranes: the external membrane is decorated with process $Ext1$ and the internal membrane is decorated with process $Ext2$. The initial state of the system is reported in Figure 3.

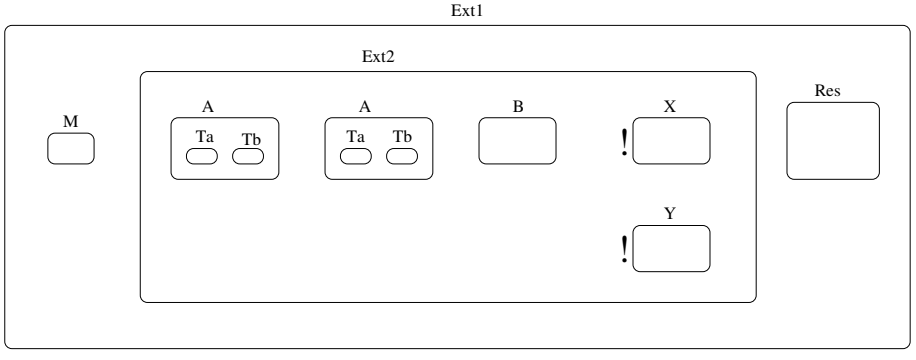


Fig. 3. The initial configuration of the system with output in the Res membrane (with $M = mate_n^{\perp}$)

The system behaves as the system presented in the previous subsection as far as the generation of new copies of C is concerned. On the other hand, when we decide to terminate (by choosing the second $mate_n$ action) then, instead of blocking the system, the continuation of process $Ext2$ (together with system $!Y(\!|\ |\!)$) permits to the nested membranes A,A' and B to perform an exocytosis. In this way, all the C membranes (as well as the terminating Ta and Tb membranes) are put in the region of the external membrane. The $Ext2$ membrane, as well as the E membranes, disappear by performing an exocytosis with the external membrane, whereas each C membrane produces a child decorated with C' by pinocytosis, and then fuses with the Res membrane.

When the computation stops, the result is represented by the number of C' membranes contained inside the Res membrane, and the structure of the system is depicted in Figure 4.

Formally, the system is defined as follows:

$$
\begin{aligned}
Ext1(\!|mate_n^{\perp}(\!|\ \ |\!)\circ\ Ext2(\!|\ & A(\!|\ Ta(\!|\ \ |\!)\ \circ\ Tb(\!|\ \ |\!)\ |\!)\ \circ \\
& A(\!|\ Ta(\!|\ \ |\!)\ \circ\ Tb(\!|\ \ |\!)\ |\!)\ \circ \\
& B(\!|\ \ |\!)\ \circ \\
& !(X(\!|\ \ |\!))\circ \\
& !(Y(\!|\ \ |\!))|\!)\ \circ \\
Res(\!|\ & \ \ |\!)\ |\!)
\end{aligned}
$$

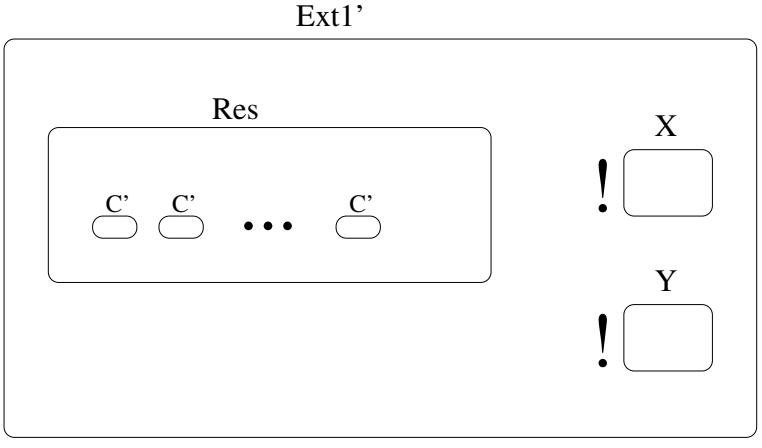


Fig. 4. The final configuration of the system with output in the *Res* membrane

The processes $Ext1$ and $Ext2$ are defined as follows:

$$Ext1 = !\circlearrowleft^{\perp}_{out}$$

$$\begin{aligned} Ext2 = {} & !mate_n.\ \circledcirc(mate^{\perp}_{a_s}).\ \circledcirc(mate^{\perp}_{a_s}).pino(mate^{\perp}_{b_s}).\circlearrowleft^{\perp}_{a_f}.\circlearrowleft^{\perp}_{a_f}. \\ & \circlearrowleft^{\perp}_{b_f}.drip(mate^{\perp}_{n}) \mid \\ & mate_n.\circlearrowleft^{\perp}_{a_e}.\circlearrowleft^{\perp}_{a_e}.\circlearrowleft^{\perp}_{b_e}.\circlearrowleft_{out} \end{aligned}$$

The definitions of the remaining systems and processes are as follows:

$$\begin{aligned} A &= mate_{a_s}.\ \circledcirc(mate^{\perp}_{a_s}).\ \circledcirc(mate^{\perp}_{b_s}).\circlearrowleft^{\perp}_{a_f}.\circlearrowleft^{\perp}_{b_f}.drip(D).\circlearrowleft_{d}.A' \mid \circlearrowleft_{a_e} \\ A' &= !mate_{a_s}.\ \circledcirc(mate^{\perp}_{a_s}).\ \circledcirc(mate^{\perp}_{b_s}).\circlearrowleft^{\perp}_{a_f}.\circlearrowleft^{\perp}_{b_f}.drip(\circlearrowleft_{a_f}) \mid \circlearrowleft_{a_e} \\ D &= \circlearrowleft^{\perp}_{d}(E).\ \circledcirc(B).drip(\circlearrowleft_{a_f}).mate^{\perp}_{x} \\ X &= mate_x.A \\ Y &= mate_y.\circlearrowleft^{\perp}_{a_e}.\circlearrowleft^{\perp}_{b_e}.\circlearrowleft^{\perp}_{out} \\ E &= !mate_{a_s}.\ \circledcirc(mate^{\perp}_{a_s}).\circlearrowleft^{\perp}_{a_f}.drip(\circlearrowleft_{a_f}) \mid mate^{\perp}_{y} \\ B &= !mate_{b_s}.\ \circledcirc(C).drip(\circlearrowleft_{b_f}) \mid \circlearrowleft_{b_e} \\ Ta &= (!mate_{a_s}.drip(\circlearrowleft_{a_f})) \mid \circlearrowleft_{out} \\ Tb &= (!mate_{b_s}.drip(\circlearrowleft_{b_f})) \mid \circlearrowleft_{out} \\ Res &= !mate^{\perp}_{res} \\ C &= \circledcirc(C').mate_{res} \end{aligned}$$

5 Final Remarks

In the last years, two branches of Natural Computing, *Membrane Computing* and *Brane Calculi* have been developed at the crossroads of Cell Biology and Computation. Both branches start from the idea of cells are capable to process and to generate information. Nonetheless, they have followed different paths.

Membrane Computing are more interested in the study of computational devices, by taking the cell as inspiration whereas Brane Calculi try to stay as close to the Biology as possible.

In a certain sense, Brane Calculi are dual to Membrane Computing, since they work with object placed *on* membranes, not with object placed in the regions surrounded by membranes. This is a key difference. In Membrane Computing, the objects represent chemicals swimming in an aqueous solution inside the membranes and membranes separate the compartments where local rules are applied. In Brane Calculi, objects are placed on membranes and they correspond to proteins embedded in the real membranes. The computation is made by membrane operations controlled by these objects.

Another notable difference between Brane Calculi and P systems is concerned with the semantics of the two formalism: 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).

In this paper we started a joint investigation of both formalisms inspired by the behavior of biological membranes. In particular, we investigate their computational power w.r.t. their ability to generate sets of numbers, and we take as a case study the set $\mathcal{L} = \{n^2 \mid n \geq 1\}$.

First we recalled the P systems presented in [11] which generates $\mathcal{L}$, then we provided a new, simplified solution. Then we move to Brane Calculi, and we tackle the problem of presenting a solution to the case study based on the simplified solution we propose for P systems. After discussing the problems which arise when moving from P systems to Brane Calculi, we present two solutions of the problem in Brane Calculi. The most relevant problem is due to the shift from the maximal parallelism semantics of P systems to the interleaving semantics of Brane Calculi: while maximal parallelism turns out to be a very powerful synchronization tool, permitting to synchronize an unbounded number of components, it seems that this form of synchronization turns out to be problematic in Brane Calculi. We solve this problem by moving from a "flat" representation of the system to a hierarchical representation, that can be easily obtained by making use of an unbounded number of membranes.

We think that the present paper could represent a first step in the comparison of the two aforementioned formalisms. As future work, we plan to investigate the possibility to compute **NP**-complete problems in polynomial time with Brane Calculi, by taking as a starting point the encouraging results on this topic obtained for P systems (see, for example, [13] and references therein).

Acknowledgement

The second author acknowledges the support by Project TIN2005-09345-C03-01 of the Ministry of Education and Science of Spain, cofinanced by FEDER funds, and by the Project of Excellence TIC-581 of the Junta de Andalucía.

References

1. L.M. Adleman. Molecular computations of solutions to combinatorial problems. *Science*, 226 (1994), 1021–1024.
2. D. Besozzi, N. Busi, G. Franco, R. Freund, Gh. Păun. Two universality results for (mem)brane systems. In *Proceedings of the Fourth Brainstorming Week on Membrane Computing, Vol. I* (M.A. Gutiérrez Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, eds.), Fénix Editora, 2006, 49–62.
3. N. Busi, R. Gorrieri. On the computation power of brane calculi. Third Workshop on Computational Methods in Systems Biology, Edinburgh, 2005.
4. L. Cardelli. Brane calculi. In *Computational Methods in Systems Biology 2004* (V. Danos, V. Schachter, eds.), LNBI **3082**, Springer-Verlag, Berlin, 2005, 257–278.
5. L. Cardelli and A.D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240, 1 (2000), 177–213.
6. L. Cardelli, Gh.Păun. An universality result for a (mem)brane calculus based on mate/drip operations. *Intern. J. Found. Computer Sci.*, 17, 1 (2006), 49–68.
7. J.H. Holland. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975.
8. W.S. McCulloch, W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5 (1943) 115–133.
9. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143.
10. Gh. Păun, M.J. Pérez-Jiménez. Recent computing models inspired from biology: DNA and membrane computing. *Theoria*, 18 (2003), 72–84.
11. Gh. Păun. *Membrane Computing – An Introduction* Springer-Verlag, Berlin, 2002.
12. A. Regev, E. M. Panina, W. Silverman, L. Cardelli, E. Shapiro. BioAmbients: An abstraction for biological compartments. *Theoretical Computer Science*, 325, 1 (2004), 141–167.
13. A. Riscos-Núñez. *Cellular Programming: Efficient Resolution of Numerical NP-Complete Problems*. Ph.D. Thesis. University of Seville, 2004.
14. P systems web page `http://psystems.disco.unimib.it/`

Computing with Genetic Gates, Proteins, and Membranes

Nadia Busi[1] and Claudio Zandron[2]

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna
Mura A. Zamboni 7, I-40127 Bologna, Italy
busi@cs.unibo.it
[2] Dipartimento di Informatica, Sistemistica e Comunicazione
Università di Milano-Bicocca
via Bicocca degli Arcimboldi 8, I-20126, Milano, Italy
zandron@disco.unimib.it

Abstract. We introduce Genetic P systems, a class of P systems with evolution rules inspired by the functioning of the genes.

The creation of new objects – representing proteins – is driven by genetic gates: a new object is produced when all the activator objects are present, and no inhibitor object is available. Activator objects are not consumed by the application of such an evolution rule. Objects disappear because of degradation: each object is equipped with a lifetime; when such a lifetime expires, the object decays.

Then, we extend the basic model with *bind and release* rules and *repressor* rules, that simulate the action of protein channels and the action of substances which connect to other objects to block their use. We provide a universality result for such a class of systems.

1 Introduction

Membrane computing is a branch of natural computing, initiated by Gheorghe Păun with the definition of P systems in [3,4,5]. The aim is to provide a formal modeling of the structure and the functioning of the cell, making use especially of automata, languages, and complexity theoretic tools.

Membrane systems (also called P systems) are based upon the notion of *membrane structure,* which is a structure composed by several cell-membranes, hierarchically embedded in a main membrane called the *skin membrane*. A plane representation of a membrane structure can be given by means of a Venn diagram, without intersected sets and with a unique superset. The membranes delimit *regions* and we associate with each region a set of *objects*, described by some symbols over an alphabet, and a set of *evolution rules*.

In the basic variant, the objects evolve according to the evolution rules, which can modify the objects to obtain new objects and send them outside the membrane or to an inner membrane. The evolution rules are applied in a maximally parallel manner: at each step, all the objects which can evolve should evolve.

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 250–265, 2006.

A computation device is obtained: we start from an initial configuration, with a certain number of objects in certain membranes, and we let the system evolve. If a computation *halts,* that is no further evolution rule can be applied, the result of the computation is defined to be the number of objects in a specified membrane (or expelled through the skin membrane). If a computation never halts (i.e., one or more object can be rewritten forever), then it provides no output.

An up-to-date bibliography of the area and other useful resources can be found at [9].

The goal of this paper is to introduce systems which mimic the functioning of the genes. The relevance of such a subject has been recently pointed out in [6]. Genetic gates work in the following way: the production of a substance is the result of the activation of a gene, when certain substances (activators) are present while other substances (inhibitors) are absent. It is important to stress the fact that the production of the object does not require that one or more objects are consumed in order to do this. Nonetheless, objects can disappear due to a decay process. For this reason, objects are marked with a lifetime, which is decreased by one at each computation step. When this value becomes equal to zero, the object disappears.

We also consider rules to simulate the action of protein on membranes to communicate objects through protein channels, by defining *bind and release* rules, and the action of certain substances which act as repressors by connecting to other objects so to block their action, by defining *repressor* rules. We show that systems with all these types of rules are universal, and we point out various questions and investigation topics for further research.

The rest of the paper is organized as follows. In Section 2 we give some basic definitions which will be used throughout the paper. In Section 3 we define Genetic P systems and in section 4 we extend the the basic class with *Bind and Release* rules and repressor rules. In Section 5 we provide an universality result for such an extended class of systems. Section 6 gives some conclusive remarks and presents various research topics.

2 Basic Definitions

In this section we provide some basic definitions that will be used throughout the paper. We start with the definition of multisets and multiset operations.

Definition 1. *Given a set S, a* finite multiset *over S is a function $m : S \to \mathbb{N}$ such that the set $dom(m) = \{s \in S \mid m(s) \neq 0\}$ is finite. The* multiplicity *of an element s in m is given by the natural number $m(s)$. The set of all finite multisets over S, denoted by $\mathcal{M}_{fin}(S)$, is ranged over by m. A multiset m such that $dom(m) = \emptyset$ is called* empty. *The empty multiset is denoted by $\emptyset$.*

Given the multiset m and m', we write $m \subseteq m'$ if $m(s) \leq m'(s)$ for all $s \in S$ while $\oplus$ denotes their multiset union*: $m \oplus m'(s) = m(s) + m'(s)$. The operator $\setminus$ denotes* multiset difference*: $(m \setminus m')(s) =$ if $m(s) \geq m'(s)$, then $m(s) - m'(s)$*

else 0. *The* scalar product, $j \cdot m$, *of a number* j *with* m, *is* $(j \cdot m)(s) = j \cdot (m(s))$. *The cardinality of a multiset is the number of occurrences of elements contained in the multiset:* $|m| = \sum_{s \in S} m(s)$.

The set of parts of a set S is defined as $\mathcal{P}(S) = \{X \mid X \subseteq S\}$.

Given a set $X \subseteq S$, with abuse of notation we use X to denote also the multiset

$$m_X(s) = \begin{cases} 1 & \text{if } s \in X \\ 0 & \text{otherwise} \end{cases}$$

The restriction to a subset of a multiset is defined as follows:

Definition 2. *Let* m *be a finite multiset over* S *and* $X \subseteq S$. *The multiset* $m|_X$ *is defined as follows: for all* $s \in S$,

$$m|_X(s) = \begin{cases} m(s) & \text{if } s \in X \\ 0 & \text{otherwise} \end{cases}$$

We provide some basic definitions on strings, cartesian products, and relations.

Definition 3. *A string over* S *is a finite (possibly empty) sequence of elements in* S. *Given a string* $u = x_1 \dots x_n$, *the length of* u *is the number of occurrences of elements contained in* u *and is defined by* $|u| = n$. *The empty string is denoted by* λ.

With S^* *we denote the set of strings over* S, *and* $u, v, w, \dots$ *range over* S. *Given* $n \geq 0$, *with* S^n *we denote the set of strings of length* n *over* S.

Given a string $u = x_1 \dots x_n$ *and* i *such that* $1 \leq i \leq n$, *with* $(u)_i$ *we denote the* i*-th element of* u, *namely,* $(u)_i = x_i$.

Given a string $u = x_1 \dots x_n$, *the multiset corresponding to* u *is defined as follows: for all* $s \in S$, $m_u(s) = |\{i \mid x_i = s, 1 \leq i \leq n\}|$. *With abuse of notation, we use* u *to denote also* m_u.

Definition 4. *With* $S \times T$ *we denote the cartesian product of sets* S *and* T, *with* $\times_n S$, $n \geq 1$, *we denote the cartesian product of* n *copies of set* S *and with* $\times_{i=1}^n S_i$ *we denote the cartesian product of sets* $S_1, \dots, S_n$, *i.e.,* $S_1 \times \dots \times S_n$. *The* i*th projection of* $(x_1, \dots, x_n) \in \times_{i=1}^n S_i$ *is defined as* $\pi_i(x) = x_i$, *and lifted to subsets* $X \subseteq \times_{i=1}^n S_i$ *as follows:* $\pi_i(X) = \{\pi_i(x) \mid x \in X\}$.

Given a binary relation R over a set S, with R^n we denote the composition of n instances or R, with R^+ we denote the transitive closure of R, and with R^* we denote the reflexive and transitive closure of R.

3 Genetic P Systems

In this section, we present the definition of Genetic P systems and the definitions which we need to describe their functioning. To this aim, we start with the definition of *membrane structure*:

Definition 5. *Given the alphabet $V = \{[,]\}$, the set MS is the least set inductively defined by the following rules:*

- $[\ ] \in MS$
- *if $\mu_1, \mu_2, \ldots, \mu_n \in MS$, $n \geq 1$, then $[\mu_1 \ldots \mu_n] \in MS$*

We define the following relation over MS: $x \sim y$ if and only if the two strings can be written in the following form: $x = [_1 \ldots [_2 \ldots]_2 \ldots [_3 \ldots]_3 \ldots]_1$ and $y = [_1 \ldots [_3 \ldots]_3 \ldots [_2 \ldots]_2 \ldots]_1$ (i.e., if two pairs of parentheses that are neighbors can be swapped together with their contents).

The set $\overline{MS}$ of membrane structures *is defined as the set of equivalence classes w.r.t. the relation $\sim^*$.*

We say that i is the father *of j (and j is a* child *of i) if the membrane j is contained in i, and no membrane exists that contains j and is contained in i.*

The partial function $father : \{1, \ldots, d\} \to \{1, \ldots, d\}$ returns the father of a membrane i, or is undefined if i is the external membrane.

The function $children : \{1, \ldots, d\} \to \mathcal{P}(\{1, \ldots, d\})$ returns the set of children of a membrane.

We call *membrane* each matching pair of parentheses appearing in the membrane structure. A membrane structure μ can be represented as a Venn diagram, in which any closed space (delimited by a membrane and by the membranes immediately inside) is called a *region* of μ.

We can give now the definition of Genetic P systems (or GP systems for short). To this aim, given a set X, we define $\mathcal{R}_X = \mathcal{P}(X) \times \mathcal{P}(X) \times X$.

Definition 6. *A* Genetic P system with timed degradation *(of degree d, with $d \geq 1$) is a construct*

$$\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, i_0)$$

where:

1. *V is a finite alphabet whose elements are called* objects*;*
2. *μ is a* membrane structure *consisting of d membranes (usually labeled with i and represented by corresponding brackets $[_i$ and $]_i$, with $1 \leq i \leq d$);*
3. *w_i^0, $1 \leq i \leq d$, are strings over $V \times (I\!N \cup \infty)$ associated with the regions $1, 2, \ldots, d$ of μ; they represent multisets of objects of the form (a, t) present in the regions of μ, where a is a symbol of the alphabet V and $t > 0$ represents the decay time of that object. The multiplicity of a pair in a region is given by the number of occurrences of this pair in the string corresponding to that region;*
4. *R_i, $1 \leq i \leq d$, are finite multisets*[1] *of* genetic gates *over V associated with the regions $1, 2, \ldots, d$ of μ; these gates are of the forms $u_{act}, \neg u_{inh} :\rightarrow (b, t)$ where $u_{act} \cap u_{inh} = \emptyset$. $u_{act} \subseteq V$ is the* positive regulation (activation)[2]*,*

[1] Here we use multisets of rules, instead of sets, because each rule can be used at most once in each computational step.

[2] We consider sets of activators, meaning that a genetic gate is never activated by more than one instance of the same protein.

$u_{inh} \subseteq V$ *is the* negative regulation (inhibition)*,* $b \in V$ *is the transcription of the gate*[3] *and* $t \in I\!N \cup \infty$ *is the* duration *of object* b*;*

5. i_0 *is a number between* 1 *and* d *and it specifies the* output *membrane of* Π*.*

We say that a gate is *unary* if $|u_{act} \oplus u_{inh}| = 1$.

The membrane structure and the multisets represented by w_i, $1 \leq i \leq d$, in Π constitute the *initial state*[4] of the system. A transition between states is governed by an application of the transcriptions specified by the genetic gates which is done in parallel; all objects, from all membranes, which *can be* the subject of local evolution (that is, that can be used to apply the rule of a gate which is not used in the same step by other objects) *have to* evolve simultaneously.

The gate $u_{act}, \neg u_{inh} :\rightarrow (b, t)$ can be activated if the region it belongs to contains enough free activators and no free inhibitors. If the gate is activated, the regulation objects (activators) in the set u_{act} are bound to such a gate, and they cannot be used for activating any other gate in the same maximal parallelism evolution step. On the contrary, if one or more free inhibitor objects are present in the region where the gate is placed, then one of these objects (non-deterministically chosen) is bound to the gate, which cannot then be activated.

In other words, the gate $u_{act}, \neg u_{inh} :\rightarrow (b, t)$ in a region containing a multiset of (not yet bound) objects m can be activated if u_{act} is contained in m and no object in u_{inh} appears in m; if the gate performs the transcription, then a new object (b, t) is produced. Note that the objects in u_{act} and u_{inh} are not consumed by the transcription operation, but will be released at the end of the operation and (if they do not disappear because of the decay process) they can be used in the next maximal parallelism evolution step. Each object starts with a decay number, which specify the number of steps after which this object disappears. The decay number is decreased after each parallel step; when it reaches the value zero, the object disappears. If the decay number of an object is equal to ∞, then the object is persistent and it never disappears.

Note that the decay number associated to an object depends on the gate that produced the object (if the object is not present in the initial system), and not on the type of the object. Hence,a system may contain two gates, say, e.g. $a :\rightarrow (b, 5)$ and $a, \neg c :\rightarrow (b, \infty)$: the first gate produces one copy of object b that decays after 5 time units, whereas the second gate produces a persistent copy of object b.[5]

We adopt the following notation for gates. The activation and inhibition sets are denoted by one of the corresponding strings, i,e, $a, b, \neg c :\rightarrow (c, 5)$ denotes the gate $\{a, b\}, \neg\{c\} :\rightarrow (c, 5)$. If either the activation or the inhibition is empty then we omit the corresponding set, i.e., $a :\rightarrow (b, 3)$ is a shorthand for the gate $\{a\}, \neg\emptyset :\rightarrow (b, 3)$. The *nullary gate* $\emptyset, \neg\emptyset :\rightarrow (b, 2)$ is written as $:\rightarrow (b, 2)$.

[3] Usually the expression of a genetic gate consists of a single protein.

[4] Here we use the term *state* instead of the classical term *configuration* because we will define a (essentially equivalent but syntactically) different notion of configuration in Section 5.

[5] We could also consider a variant of GP systems where the decaying time is a function of the type of the object, i.e., all the objects b that are produced in the system will have the same decaying time. We plan to devote future investigation to this variant.

3.1 Partial Configurations, Reaction Relation, and Maximal Parallelism Step

Once defined GP systems, we are ready to describe their functioning. Hence, we give now the definitions for partial configuration, configuration, reaction relation, and heating and decaying function.

Definition 7. *Let $\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, i_0)$ be a GP system.*

A partial configuration *of Π is a tuple $(w_1, R_1, \bar{w}_1, \bar{R}_1), \ldots, (w_d, R_d, \bar{w}_d, \bar{R}_d) \in \times_d((V \times I\!N) \times \mathcal{R}_V \times (V \times I\!N) \times \mathcal{R}_V)$.*

We use $\times_{i=1}^d (w_i, R_i \bar{w}_i, \bar{R}_i)$ to denote the partial configuration above.

The set of partial configurations of Π is denoted by $Conf_\Pi$. We use γ, γ', γ_1, ... to range over $Conf_\Pi$.

$w_1, \ldots, w_d$ represent the active multisets, whereas $\bar{w}_1, \ldots, \bar{w}_d$ represent the frozen (already used) multisets, $R_1, \ldots, R_d$ represent the active gates, while $\bar{R}_1, \ldots, \bar{R}_d$ represent the frozen (already used) gates.

A *configuration* is a partial configuration containing no frozen objects; configurations represent the states reached after the execution of a maximal parallelism computation step.

Definition 8. *Let $\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, i_0)$ be a GP system.*

A configuration *of Π is a partial configuration $\times_{i=1}^d (w_i, R_i, \bar{w}_i, \bar{R}_i)$ satisfying the following: $\bar{w}_i = \emptyset$ and $\bar{R}_i = \emptyset$ for $i = 1, \ldots, d$.*

The initial configuration *of Π is the configuration $\times_{i=1}^d (w_i^0, R_i, \emptyset, \emptyset)$.*

The activation of a genetic gate is formalized by the notion of reaction relation. In order to give a formal definition we need the function $obj : (V \times I\!N)^* \to V^*$, defined as follows. Assume that $(a, t) \in (V \times (I\!N \cup \infty))$ and $w \subseteq (V \times (I\!N \cup \infty))^*$. Then, $obj(\lambda) = \lambda$ and $obj((a, t)w) = a\ obj(w)$.

We also need to define a function DecrTime which is used to decrement the decay time of objects, destroying the objects which reached their time limit.

Definition 9. *The function $DecrTime : (V \times I\!N)^* \to (V \times I\!N)^*$ is defined as follows:*

$DecrTime(\lambda) = \lambda$

and

$$DecrTime((a, t)w) = \begin{cases} (a, t-1)DecrTime(w) & \text{if } t > 1 \\ DecrTime(w) & \text{if } t = 1 \end{cases}$$

We are now ready to give the notion of a *reaction relation*.

Definition 10. *Let $\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, i_0)$ be a GP system.*

The reaction relation $\mapsto$ *over $Conf_\Pi \times Conf_\Pi$ is defined as follows:*

$\times_{i=1}^d (w_i, R_i, \bar{w}_i, \bar{R}_i) \mapsto \times_{i=1}^d (w_i', R_i', \bar{w}_i', \bar{R}_i')$ iff there exist k, with $1 \leq k \leq d$ and $u_{act}, \neg u_{inh} :\to (b, t) \in R_k$ such that

- $R'_k = R_k \setminus (u_{act}, \neg u_{inh} :\rightarrow (b,t))$
- $\bar{R}'_k = \bar{R}_k \oplus (u_{act}, \neg u_{inh} :\rightarrow (b,t))$
- $\forall i : 1 \leq i \leq d$ *and* $i \neq k$ *implies* $w'_i = w_i$, $\bar{w}'_i = \bar{w}_i$, $R'_i = R_i$ *and* $\bar{R}'_i = \bar{R}_i$
- *if* $u_{inh} \cap dom(obj(w_k)) = \emptyset$ *and* $\exists w_{act} \subseteq w_k$ *such that*[6] $obj(w_{act}) = u_{act}$ *then*
  - $w'_k = w_k \setminus w_{act}$
  - $\bar{w}'_k = \bar{w}_k \oplus \{(b,t)\} \oplus DecrTime(w_{act})$
- *if* $\exists (s,t) \in dom(w_k)$ *such that* $s \in u_{inh}$ *then*
  - $w'_k = (w_k) \setminus (s,t)$
  - $\bar{w}'_k = \bar{w}_k \oplus DecrTime((s,t))$

Definition 11. *The function* $heat\&decay : Conf_\Pi \rightarrow \mathcal{P}(Conf_\Pi)$ *is then defined as follows:*

$heat\&decay(\times_{i=1}^{d}(w_i, R_i, \bar{w}_i, \bar{R}_i)) = \times_{i=1}^{d}((DecrTime(w_i)) \oplus \bar{w}_i), R_i \oplus \bar{R}_i, \emptyset, \emptyset)$

Now we are ready to define the maximal parallelism computational step $\Rightarrow$:

Definition 12. *Let* $\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, i_0)$ *be a GP system.*

The maximal parallelism computational step $\Rightarrow$ *over (nonpartial) configurations of* Π *is defined as follows:* $\gamma_1 \Rightarrow \gamma_2$ *iff there exists a partial configuration* γ' *such that* $\gamma_1 \mapsto^+ \gamma'$, $\gamma' \not\mapsto$ *and* $\gamma_2 = heat\&decay(\gamma')$.

4 Genetic P Systems with Bind&Release and Repressor Rules

The use of genetic gates alone is quite restrictive. For instance, no communication of objects is possible through the membranes, a feature which is fundamental in the basic variant of P systems. In fact, without communication the system would not act as a whole unit, but instead as a collection of separate systems or processes of various types, without interaction.

In order to enrich the model described so far, we consider also two other types of rules which mimic two different important cellular reactions.

The first type of rules we consider (Bind&Release), mimics the communication of objects through a protein channel. Two (multisets of) substances are *bound* to both sides of a membrane. Then, by means of a channel in the membrane, they pass in opposite directions through the membrane itself, exchanging in this way their position. Finally, they can be *released* in their (new) region.

The second type of rules we consider (Repressor) mimics the action of certain substances that act to deactivate other substances present in the cell. When such a substance (a repressor) get in contact with another object, it creates a bond which cannot be destroyed. The object (and the repressor substance) cannot be used anymore for any other reaction. We notice that the repressor can create such a bond in any region of the cell. For this reason, the set of repressor rules will be valid for the whole system (i.e., we will not define different set of repressor rules for each region).

[6] The symbol = should be intended here as working on multisets.

We provide the definition of Genetic P systems extended with Bind&Release and Repressor rules:

Given a set X, we define $\mathcal{R}_X = \mathcal{P}(X) \times \mathcal{P}(X) \times X$ and $\mathcal{B}R_X = \mathcal{P}(X) \times \mathcal{P}(X) \times \mathcal{P}(X) \times \mathcal{P}(X)$

Definition 13. *A* Genetic P system with timed degradation, Bind and Release actions and repressor rules *(of degree d, with $d \geq 1$), or* G^+P system *for short, is a construct*

$$\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, BR_1, \ldots, BR_d, R_s, i_0)$$

where

1. *V, μ, i_0, and w_i^0, R_i, for $1 \leq i \leq d$, are defined as in Definition 6.*
2. *BR_i, $1 \leq i \leq d$, are finite multisets of* Bind and Release rules *over V associated with the regions $1, 2, \ldots, d$ of μ; these rules are of the forms $u[v] \rightarrow v[u]$ where $u, v \in V^*$, and $|uv| > 0$. The* weight *of a Bind and Release rule $u[v] \rightarrow v[u]$ is $|u| + |v|$.*
3. *R_s is a finite multiset of* repressor rules*; these rules are associated with the system (and not to each region), and they are of the form $a, b \rightarrow a\&b$ where $a, b \in V$.*

Besides evolution driven by the application of transcriptions specified by genetic gates and object degradation, evolution steps in G^+P systems are also concerned with object migration through membranes and proteins repression.

Objects can be moved through membranes using bind and release operations. If outside a region i is present a multiset u of objects in $(V \times I\!N)$ and inside i a multiset v of objects in $(V \times I\!N)$, then a rule $u[v] \rightarrow v[u]$ in BR_i can be activated, moving the multisets u and v outside and inside region i, respectively.

Finally some objects can act as repressor objects, by means of repressor rules R_s. Such a rules of the form $a, b \rightarrow a\&b$ is activated when a repressor object b is present, thus binding to an object a and creating a new object which cannot be used anymore with any other rule.

4.1 Partial Configurations, Reaction Relation, and Maximal Parallelism Step

As we did for the basic case, we give now the definitions for partial configuration, configuration, reaction relation, and heating and decaying function for G^+P systems.

Definition 14. *Let*

$$\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, BR_1, \ldots, BR_d, R_s, i_0)$$

be a G^+P system.

A partial configuration *of Π is a tuple*
$(w_1, R_1, BR_1, \bar{w}_1, \bar{R}_1, \overline{BR}_1), \ldots, (w_d, R_d, BR_d, \bar{w}_d, \bar{R}_d, \overline{BR}_d)$
$\in \times_d((V \times I\!N) \times \mathcal{R}_V \times \mathcal{B}R_V \times (V \times I\!N) \times \mathcal{R}_V \times \mathcal{B}R_V)$.

We use $\times_{i=1}^{d}(w_i, R_i, BR_i, \bar{w}_i, \bar{R}_i, \overline{BR}_i)$ *to denote the partial configuration above. The set of partial configurations of* Π *is denoted by* $Conf_\Pi$*. We use* $\gamma, \gamma', \gamma_1, \ldots$ *to range over* $Conf_\Pi$*.*

$w_1, \ldots, w_d$ represent the active multisets, $\bar{w}_1, \ldots, \bar{w}_d$ represent the frozen (already used) multisets, $R_1, \ldots, R_d$ represent the active gate rules, $\bar{R}_1, \ldots, \bar{R}_d$ represent the frozen (already used) gate rules, $BR_1, \ldots, BR_d$ represent the active Bind&Release rules, $\bar{R}_1, \ldots, \bar{R}_d$ represent the frozen (already used) Bind& Release rules.

A *configuration* is a partial configuration containing no frozen objects; configurations represent the states reached after the execution of a maximal parallelism computation step.

Definition 15. *Let*

$$\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, BR_1, \ldots, BR_d, R_s, i_0)$$

be a G^+P *system.*

A configuration *of* Π *is a partial configuration* $\times_{i=1}^{d}(w_i, R_i, BR_i, \bar{w}_i, \bar{R}_i, \overline{BR}_i)$ *satisfying the following:* $\bar{w}_i = \emptyset$*,* $\bar{R}_i = \emptyset$ *and* $\overline{BR}_i = \emptyset$ *for* $i = 1, \ldots, d$*.*

The initial configuration *of* Π *is the configuration* $\times_{i=1}^{d}(w_i^0, R_i, BR_i, \emptyset, \emptyset, \emptyset)$*.*

The activation of a genetic gate is formalized by the notion of reaction relation. In order to give a formal definition we need the function $obj : (V \times I\!N)^* \to V^*$, defined as follows. Assume that $(a, t) \in (V \times (I\!N \cup \infty))$ and $w \subseteq (V \times (I\!N \cup \infty))^*$. Then, $obj(\lambda) = \lambda$ and $obj((a, t)w) = a\ obj(w)$.

We also need to define a function DecrTime which is used to decrement the time index used objects, destroying the objects which reached their time limits. The function $DecrTime : (V \times I\!N)^* \to (V \times I\!N)^*$ is defined as follows:

Definition 16. $DecrTime(\lambda) = \lambda$
and

$$DecrTime((a, t)w) = \begin{cases} (a, t-1)DecrTime(w) & \text{if } t > 1 \\ DecrTime(w) & \text{if } t = 1 \end{cases}$$

We are now ready to give the notion of *reaction relation.*

Definition 17. *Let*

$$\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, BR_1, \ldots, BR_d, R_s, i_0)$$

be a G^+P *system.*

The reaction relation $\mapsto$ *over* $Conf_\Pi \times Conf_\Pi$ *is defined as follows:*

$\times_{i=1}^{d}(w_i, R_i, BR_i, \bar{w}_i, \bar{R}_i, \overline{BR}_i) \mapsto \times_{i=1}^{d}(w_i', R_i', BR_i', \bar{w}_i', \bar{R}_i', \overline{BR}_i')$ *iff there exist* k*, with* $1 \leq k \leq d$ *and a rule* R *in* $R_k \cup BR_k \cup R_s$ *such that*

CASE 1: IF $R : u_{act}, \neg u_{inh} :\rightarrow (b, t) \in R_k$*, THEN*

- $R_k' = R_k \setminus (u_{act}, \neg u_{inh} :\rightarrow (b, t))$
- $\bar{R}_k' = \bar{R}_k \oplus (u_{act}, \neg u_{inh} :\rightarrow (b, t))$

- $\forall i : 1 \leq i \leq d$ *and* $i \neq k$ *implies* $w'_i = w_i$, $\bar{w}'_i = \bar{w}_i$, $R'_i = R_i$ *and* $\bar{R}'_i = \bar{R}_i$
- *if* $u_{inh} \cap dom(obj(w_k)) = \emptyset$ *and* $\exists w_{act} \subseteq w_k$ *such that* $obj(w_{act}) = u_{act}$ *then*
  - $w'_k = w_k \setminus w_{act}$
  - $\bar{w}'_k = \bar{w}_k \oplus \{(b,t)\} \oplus DecrTime(w_{act})$
- *if* $\exists (s,t) \in dom(w_k)$ *such that* $s \in u_{inh}$ *then*
- $w'_k = (w_k) \setminus (s,t)$
- $\bar{w}'_k = \bar{w}_k \oplus DecrTime((s,t))$

CASE 2: IF $R : u[v] \rightarrow v[u] \in BR_k$ *THEN*

- $\exists U_{br} \subseteq w_{father(k)}$ *and* $\exists V_{br} \subseteq w_k$ *such that* $u = obj(U_{br})$ *and* $v = obj(V_{br})$
- $\forall i : 1 \leq i \leq d$, $i \neq k$ *and* $i \neq father(k)$ *implies* $w'_i = w_i$, $\bar{w}'_i = \bar{w}_i$, $BR'_i = BR_i$ *and* $\overline{BR}'_i = \overline{BR}_i$
- $BR'_{father(k)} = BR_{father(k)}$
- $\overline{BR}'_{father(k)} = \overline{BR}_{father(k)}$
- $w'_{father(k)} = w_{father(k)} \setminus U_{br}$
- $\bar{w}'_{father(k)} = \bar{w}_{father(k)} \oplus DecrTime(V_{br})$
- $BR'_k = BR_k \setminus (u[v] \rightarrow v[u])$
- $\overline{BR}'_k = \overline{BR}_k \oplus (u[v] \rightarrow v[u])$
- $w'_k = (w_k) \setminus V_{br}$
- $\bar{w}'_k = \bar{w}_k \oplus DecrTime(U_{br})$

CASE 3: $R : a, b \rightarrow a\&b \in R_s$

- $\exists (a,t_1), (b,t_2) \in (V \times I\!N)$ *and* $(a,t_1), (b,t_2) \in w_k$
- $\forall i : 1 \leq i \leq d, i \neq k$ $w'_i = w_i$ *and* $\bar{w}'_i = \bar{w}_i$
- $\forall i : 1 \leq i \leq d$, $R'_i = R_i$ *and* $\bar{R}'_i = \bar{R}'_i$
- $\forall i : 1 \leq i \leq d$, $BR'_i = BR_i$ *and* $\overline{BR}'_i = \overline{BR}'_i$
- $w'_k = w_k \setminus (a,t_1) \setminus (b,t_2)$
- $\bar{w}'_k = \bar{w}_k \oplus (a\&b, min(t_1,t_2))$

Definition 18. *The function* $heat\&decay : Conf_{\Pi} \rightarrow \mathcal{P}(Conf_{\Pi})$ *is defined as follows:*

$heat\&decay(\times_{i=1}^{d}(w_i, R_i, BR_i, \bar{w}_i, \bar{R}_i, \overline{BR}_i)) =$
$\times_{i=1}^{d}(DecrTime(w_i) \oplus \bar{w}_i), R_i \oplus \bar{R}_i, BR_i \oplus \overline{BR}_i, \emptyset, \emptyset, \emptyset)$

Now we are ready to define the maximal parallelism computational step $\Rightarrow$:

Definition 19. *Let* $\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, BR_1, \ldots, BR_d, R_s, i_0)$ *be a* G^+P *system.*

The maximal parallelism computational step $\Rightarrow$ *over (non-partial) configurations of* Π *is defined as follows:* $\gamma_1 \Rightarrow \gamma_2$ *iff there exists a partial configuration* γ' *such that* $\gamma_1 \mapsto^+ \gamma'$, $\gamma' \not\mapsto$ *and* $\gamma_2 = heat\&decay(\gamma')$.

5 Turing Equivalence of G^+P Systems

In this section we show that G^+P systems with Bind and Release rules of weight one are Turing powerful. The result is proved by showing how to model Random Access Machines (RAMs) [8], a well known Turing powerful formalism.

We start recalling the definition of RAMs.

5.1 Random Access Machines

RAMs are a computational model based on finite programs acting on a finite set of registers. More precisely, a RAM R is composed of the registers $r_1, \ldots, r_n$, that can hold arbitrary large natural numbers, and by a sequence of indexed instructions $(1 : I_1), \ldots, (m : I_m)$. In [2] it is shown that the following two instructions are sufficient to model every recursive function:

- $(i : Succ(r_j))$: adds 1 to the contents of register r_j and goes to the next instruction;
- $(i : DecJump(r_j, s))$: if the contents of the register r_j is not zero, then decreases it by 1 and goes to the next instruction, otherwise jumps to the instruction s.

The computation starts from the first instruction and it continues by executing the other instructions in sequence, unless a jump instruction is encountered. The execution stops when an instruction number higher than the length of the program is reached.

A state of a RAM is modeled by $(i, c_1, \ldots, c_n)$, where i is the program counter indicating the next instruction to be executed, and $c_1, \ldots, c_n$ are the current contents of the registers $r_1, \ldots, r_n$, respectively. We use the notation $(i, c_1, \ldots, c_n) \rightarrow_R (i', c'_1, \ldots, c'_n)$ to denote that the state of the RAM R changes from $(i, c_1, \ldots, c_n)$ to $(i', c'_1, \ldots, c'_n)$, as a consequence of the execution of the i-th instruction.

A state $(i, c_1, \ldots, c_n)$ is *terminated* if the program counter i is strictly greater than the number of instructions m. We say that a RAM R *terminates* if its computation reaches a terminated state. The *output* of the RAM is the contents of register r_1 in the terminated state of the RAM (if such a state exists).

5.2 Encoding RAMS in G^+P Systems

In this section we show how to model RAMs in G^+P systems. Given a RAM with n registers, the system is composed by an external membrane, containing n children membranes, each one representing one register: $[_0[_1\ ]_1 \ldots [_n\ ]_n]_0$ (to simplify the notation, we label the external membrane with 0 instead of 1). The fact that register r_i contains value c_i is represented by the presence of c_i copies of object (r_i, ∞) in the membrane i. The instructions are encoded by genetic gates. The presence of object p_i in some part of the system represents the fact that the program counter contains the value i (i.e., the next instruction to be executed is the ith). At the beginning of the computation, an object $(p_i, 2)$ is in the external membrane. All the objects representing the program counter will be produced with duration 2.

As the output of the RAM is the contents of register r_1 in the terminated state of the RAM, the output of the RAM encoding is the number of occurrences of object (r_1, ∞) in membrane 1.

Usually, when providing a RAM encoding of a P system, the output of the RAM encoding is taken in (one of the) halting configurations of the encoding.

When considering GP systems, we note that it is not trivial to define what is a halting configuration. Take, e.g., the system with a negative gate $\neg a :\rightarrow (b, t)$, reaching a configuration containing only a persistent object (a, ∞): according to the reaction relation, this system never terminates. Actually, no real computation is performed, but what happens is that the inhibitor protein (a, ∞) is attacked to the negative regulation part of the gene.

Hence, here we adopt a different "termination" condition for GP systems, quite similar to the acceptance condition of automata with final states. Namely, we consider a computation to be successfully terminated if a configuration is reached which contains a distinguished persistent object (end, ∞) in the external membrane. The definition of other suitable notions of termination for GP systems is left for future investigation.

We provide a RAM encoding which satisfies the following condition: the RAM terminates with output k if and only if the encoding of the RAM reaches a configuration containing the object (end, ∞) in the membrane 0, and containing exactly k occurrences of object (r_1, ∞) in membrane 1.

We consider RAMs that satisfy the following constraints:

1. If the RAM has m instructions, then all the jumps to addresses higher than m are jumps to the address $m + 1$.
2. The "self-loops" on DecJump instructions – i.e., instruction of the kind $(i : DecJump(r_j, i))$ – are forbidden.
3. The instruction following a DecJump (either if the decrement or if the jump is performed) is an increment.

Such constraints are not restrictive, as for any RAM not satisfying the constraints it is possible to construct an equivalent RAM (i.e., a RAM computing the same function) which satisfies the constraints above.

Consider a RAM with m instructions and n registers.

The first constraint can be easily satisfied by replacing each jump to an address higher than m to a jump to the address $m + 1$.

The second constraint can be satisfied by adding to the RAM a new register r_{n+1} that always contains the value zero, and by replacing each instruction $(i : DecJump(r_j, i))$ with a pair of instructions $(i : DecJump(r_j, i + 1))$ and $(i + 1 : DecJump(r_{n+1}, i)$. This means that the instructions following the ith instruction are shifted with one position. More in detail, for all $h : i+1 \leq h \leq m$ we replace h with $h + 1$ in all the labels of the program, as well as in all the labels occurring in the jump instructions of the program.

The third constraint can be satisfied by adding a new register r_{n+2} – that will ever be incremented and never tested – and by replacing each instruction that can be reached after performing a $DecrJump$ instruction with the instruction $Succ(r_{n+2})$, and by shifting accordingly the other instructions.

If the RAM has m instructions, then the following gate belongs to membrane 0:

$p_{m+1} :\rightarrow (end, \infty)$

This rule permits the system to signal termination when the instruction p_{m+1} is reached. (Actually, as we will see in the following, two instances of (end, ∞) are produced, but this is not a problem.)

If the ith instruction is $(i : Succ(r_j))$, then the following sequence of rules of membrane 0 is executed:

step 1: $p_i, \neg r_j :\rightarrow (r_j, \infty)$
step 2: $p_i, r_j, \neg p_{i+1} :\rightarrow (p_{i+1}, 2)$
step 3: $r_j[\ ]_j \rightarrow [r_j]_j$

If object r_j enters membrane j before the object p_{i+1} is created, no new program counter $i+1$ will be created and the system will either stop in a failed computation or diverge without reaching a configuration with object *end*. As the program counters have duration equal to 2, at step 3 the object p_i decays.

If the i-th instruction is $(i : DecJump(r_j, s))$ then the following sequence of rules is executed:

step 1: $p_i[\ ]_{l_j} :\rightarrow [p_i]_{l_j}$ (in membrane j)
If the contents of register r_j is zero (no occurrences of r_j in membrane j):
step 2: $p_i, \neg r_j :\rightarrow (p_s, 3)$ (in membrane j)
step 3: $[p_s]_j \rightarrow p_s\ [\ ]_j$ (in membrane j)

After step 2 the object p_i decays. If object p_i erroneously exits the membrane, then p_i decays just after exiting, and the system reaches a failed computation (or will diverge).

If the contents of register r_j is greater than zero:

step 2: $p_i, r_j, \neg dec_{i,j} :\rightarrow decr_{i,j}$ (in membrane j)
step 3: $r_j, decr_{i,j} \rightarrow (p_{i+1}, 3)$ (in membrane j)
step 4(1): $r_j, decr_{i,j} \rightarrow r_j \& decr_{i,j}$ (in membrane j)
step 4(2): $[p_{i+1}]_{l_j} \rightarrow p_{i+1}\ [\ ]_{l_j}$ (in membrane j)

After step 2 the object p_i decays. Steps 4(1) and 4(2) are executed in the same maximal parallelism step. If the rule at step 4(1) takes place before step 3 (i.e., the repressor bounds to r_j before that p_{i+1} is created), then no new program counter is created and the system reaches a failed configuration (or will diverge).

The formal definition of the encoding of a RAM R with m instructions and n registers, whose registers $r_1, \ldots, r_n$ contain values $c_1, \ldots, c_n$ is reported in Table 1.

If some of the registers contain a value greater than zero when the RAM terminates, then the system reaches a configuration containing the *end* object, but because of gates $p_i, \neg r_j :\rightarrow (p_s, 3)$ the system will never terminate. To obtain an encoding that guarantees that the configurations containing the *end* object can perform no further computation, we could add to the RAM a further register r_{n+1}, that will never be decreased, and consider only RAMs that terminate with all registers empty but r_{n+1}, and the result is contained in register r_{n+1}. If we provide a slight variation of the encoding, where membrane $n+1$ contains no gates (as register r_{n+1} can only be increased), then the above requirement is fulfilled.

Table 1. The G^+P system encoding a RAM R

$\Pi(R) = (V, \mu, w_0^0, \ldots, w_d^0, R_0, \ldots, R_d, BR_0, \ldots, BR_d, R_s, i_0)$

$V = \{p_i \mid 1 \leq i \leq m+1\} \cup \{r_i \mid 1 \leq i \leq n\} \cup$
$\{decr_{i,j} \mid 1 \leq i \leq m+1 \wedge 1 \leq j \leq n\} \cup \{end\}$

$\mu = [_0[_1\ ]_1 \ldots [_n\ ]_n]_0$

$w_0^0 = (p_1, 2)$

$|w_j^0| = c_j$ and $(w_j^0)_i = (r_j, \infty)$ $j = 1, \ldots, n$ and $i = 1, \ldots, c_j$

$R_0 =$
$\{p_{m+1} :\rightarrow (end, \infty)\} \cup$
$\{p_i, r_j, \neg p_{i+1} :\rightarrow (p_{i+1}, 2) \mid$ the ith instr. is $(i : Succ(r_j)),\ i = 1, \ldots, m\} \cup$
$\{p_i, r_j, \neg p_{i+1} :\rightarrow (p_{i+1}, 2) \mid$ the ith instr. is $(i : Succ(r_j)),\ i = 1, \ldots, m\}$

$R_j =$
$\{p_i, \neg r_j :\rightarrow (p_s, 3) \mid$ the ith instr. is $(i : DecrJump(r_j, s))\} \cup$
$\{p_i, r_j, \neg dec_{i,j} :\rightarrow decr_{i,j} \mid$ the ith instr. is $(i : DecrJump(r_j, s))\} \cup$
$\{r_j, decr_{i,j} \rightarrow (p_{i+1}, 3) \mid$ the ith instr. is $(i : DecrJump(r_j, s))\}$

$BR_0 = \{r_j[\ ] \rightarrow [r_j] \mid 1 \leq j \leq n\}$

$BR_j =$
$\{r_j[\ ] \rightarrow [r_j]\} \cup$
$\{p_i[\ ] \rightarrow [p_i] \mid$ the ith instr. is $(i : DecrJump(r_j, s)),\ i = 1, \ldots, m\} \cup$
$\{[p_i] \rightarrow p_i[\ ] \mid$ the ith instr. is $(i : Succ(r_j)),\ i = 1, \ldots, m\}$

$R_s = \{r_j, decr_{i,j} \rightarrow r_j \& decr_{i,j} \mid 1 \leq j \leq n \wedge 1 \leq i \leq m+1\}$

$i_0 = 1$

Another feature of the encoding is the fact that, if an erroneous action is performed, then the system can reach a failed configuration (i.e., a deadlocked configuration that does not contain the *end* object). It is possible to produce an encoding that diverges when an erroneous action is performed, by adding to the membrane 0 the gate $\neg end :\rightarrow loop$. However, in such a case, the configuration containing the *end* object is no longer terminated. A possible solution could be to signal termination by emitting the *end* object outside the external membrane.

In this section, we only use a restricted version of the Bind and Release rules, namely, rules with weight 1. We claim that, by using cooperative symport or antiport rules in combination with very simple genetic gates permitting to generate as many copies as you want of any object, Turing equivalence can be obtained

as an easy consequence of the results recalled in [7]. We stress the fact that we use Bind and Release rules of weight 1 to get universality, as symport rules of weight 2 (or alternatively antiport rules with one object entering the membrane and one object exiting the membrane) are already universal, without taking into account genetic gates.

We proved Turing equivalence of G^+P systems with Bind and Release rules of weight one and suppressor rules. We started some investigation on the expressiveness of more restricted versions of G^+P systems.

We conjecture that in G^+P systems with only positive gates and with Bind and Release rules of weight 1 (and without repressor rules) it is possible to decide if a system can reach a configuration containing a *end* object. This result could be proved by using the set saturation methods for well-structured transition systems defined in [1]. A consequence of this conjecture is the fact that such a class of systems is not Turing equivalent, according to the encoding rules defined above.

If we consider systems with both positive and negative gates and with persistent objects (i.e., objects with an infinite duration) only (and without bind and release rules and without repressor rules), we conjecture that the set of configurations of the system with the maximal parallelism rule is a finite state machine, hence most of the behavioral properties can be decided.

6 Conclusions

We have presented Genetic P systems, a new class of P systems where objects can be produced by means of evolution rules which are inspired from the functioning of the genes: a gene is activated (producing a new object), when certain substances (activators) are present while other substances (inhibitors) are absent.

We have also considered rules that mimic the action of proteins on membranes to communicate objects through protein channels, and rules simulating the action of repressor substances. We showed that systems with all these types of rules are universal.

Many investigations and research directions can be explored.

For instance, we can consider different kind of genetic gates, where more objects can be created at the same time by a single activation of the gate, or where the inhibition requires the presence of all inhibiting substances.

Also genetic gates where both inhibitors and activators can be attached to the gate at the same time can be considered.

In what concern the decaying process of the objects, we could also consider a non–deterministic decay process: at each parallel evolution step some objects are non–deterministically chosen to be eliminated from the set of objects in the system.

Various questions already investigated for "classic" P systems, could be investigated also for the systems defined in this paper, such as, for example, decidability, computational power, comparison with other formalisms.

We also think that such a model would be useful to be used in the systems biology area, to simulate various biological cell processes.

References

1. A. Finkel, Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256:63–92, 2001.
2. M.L. Minsky. Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs, 1967.
3. G. Păun. Computing with membranes: an introduction. *Bull. EATCS 67*, 1999.
4. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
5. G. Păun. *Membrane Computing. An Introduction.* Springer, 2002.
6. G. Păun. 2006 research topics in membrane computing. *Proc. Fourth Brainstorming Week on Membrane Computing*, Felix Editora, Sevilla, 2006.
7. Y. Rogozhin, A. Alhazov, R. Freund, Computational power of symport/antiport: History, advances, and open problems. *Proc 6th International Workshop on Membrane Computing (WMC6)*, LNCS 3850, Springer, 2006.
8. J.C. Shepherdson, J.E. Sturgis. Computability of recursive functions. *Journal of the ACM*, 10:217–255, 1963.
9. P Systems webpage. `http://psystems.disco.unimib.it.`

Classifying States of a Finite Markov Chain with Membrane Computing

Mónica Cardona[1], M. Angels Colomer[1],
Mario J. Pérez-Jiménez[2], and Alba Zaragoza[1]

[1] Department of Mathematics, University of Lleida
Avda. Alcalde Rovira Roure, 191. 25198 LLeida, Spain
{mcardona,colomer,alba}@matematica.udl.es

[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
marper@us.es

Abstract. In this paper we present a method to classify the states of a finite Markov chain through membrane computing. A specific P system with external output is designed for each boolean matrix associated with a finite Markov chain. The computation of the system allows us to decide the convergence of the process because it determines in the environment the classification of the states (recurrent, absorbent, and transient) as well as the periods of states. The amount of resources required in the construction is polynomial in the number of states of the Markov chain.

1 Introduction

Markov chains constitute an important type of stochastic processes characterized by their evolution along determinate values (called states of the process) over time. These chains represent observations of physic systems whose evolution at a future time, conditioned on their present and past values, depends only on their present value. Thus, the Markov chain loses the *memory* of its starting state.

In order to study the evolution in time of a Markov chain as well as the existence of the stationary distribution it is necessary to classify its states. This classification depends on the path structure of the chain.

In this work this problem is approached within the framework of the cellular computing with membranes. The amount of resources that we use is polynomial in the number of states. This subject has been also treated in terms of DNA computing ([1]), based on a mathematical proposition of existence rather than on the classical definition of the period of a state. This is due to the fact that DNA computing is good in detecting the existence, but it has difficulties in obtaining numerical quantifications.

The paper is structured as follows. In the next section, basic concepts concerning Markov chains and P systems are introduced. In Section 3 a semi–uniform

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 266–278, 2006.

solution to the problem of classifying the states of a Markov chain in the framework of membrane computing is presented. Moreover, a formal verification of the system is given, and the run time and the resources required in the description of the system are analyzed.

2 Preliminaries

2.1 Markov Chains

Markov chains are a class of random processes exhibiting a certain *memoryless property* and providing a fundamental ingredient in the study of randomized algorithms. Their study is one of the main areas in modern probability theory.

A Markov process is a stochastic process that has a limited form of historical dependency. Let $\{X(t) : t \in \tau\}$ be a stochastic process defined on the parameter τ. We will think of τ in terms of time and the values that $X(t)$ can assume are called *states* which are elements of a *state space* S. In the case when the set τ is *discrete* and the set S is *finite*, the Markov process is called a *discrete–time finite Markov chain*. We consider this kind of Markov chains because computer programs work in discrete steps and computers work with a finite amount of resources and have a finite number of states.

More formally, a *finite Markov chain* is a sequence $\{X_t : t \in \mathbf{N}\}$ of random variables verifying the following (Markov) property:

$$P(X_{t+1} = j/X_0 = i_0, X_1 = i_1, \ldots, X_t = i_t) = P(X_{t+1} = j/X_t = i_t).$$

That is, the value of X_{t+1} conditioned on the value of X_t, is independent of the values of random variables X_m for $m < t$.

We suppose that the state space of the chain, S, is the (finite) set of nonnegative integers $\{e_1, \ldots, e_k\}$ (whose elements are called *states* or *results*), and the chain is characterized by its evolution among these states over time.

Hence, a finite Markov chain $\{X_t : t \in \mathbf{N}\}$ provides a random process by a change of states or results $e_1, \ldots, e_k$ in certain instants of discrete times $t \in \mathbf{N}$, and where the result of each event only depends on the result of the previous event. So, such a Markov chain is characterized by the conditional distribution

$$p_{ij}(t) = P(X_t = e_j/X_{t-1} = e_i), \text{ for all } t \geq 1,$$

which is called the *transition probability* of the process, providing one–step transition probability.

We say that a finite Markov chain is *time homogeneous* or it has *stationary transition probabilities* if the dependence between consecutive states does not change, that is, $P(X_n = e_j/X_{n-1} = e_i) = P(X_{n+m} = e_j/X_{n+m-1} = e_i)$, for all $n, m \in \mathbf{N}$, $e_i, e_j \in S$. In this case, we write the transition probability as $p_{ij} = P(X_n = e_j/X_{n-1} = e_i)$. These probabilities form a stochastic matrix $P = (p_{ij})$ with $\sum_{j=1}^{k} p_{ij} = 1, \forall i \in \{1, \ldots, k\}$, called *transition matrix*. If $p_{ij} \neq 0$ then we say that the transition from the state e_i to state e_j is possible.

The Markov property allows us to write an expression for the probability of a transition in one, two, three or more steps. For $n = 1$ this probability is simply p_{ij} and is given by the position (i, j) of the transition matrix P. For $n = 2$ the probability that the chain is in state e_j at step 2 is $p_{ij}^{(2)} = \sum_{r=1}^{k} p_{ir} p_{rj}$, and is the position (i, j) of the matrix P^2. In general, the probability that the process is in state e_j n steps after being in state e_i is given by $p_{ij}^{(n)} = \sum_{r=1}^{k} p_{ir}^{(m)} p_{rj}^{(n-m)}, 0 \leq m \leq n$, and is the position (i, j) of the matrix $P^n = P^m P^{n-m}$ (by the rules for matrix multiplication).

The conditions

$$\begin{cases} p_{ij}^{(1)} = p_{ij}, \\ p_{ij}^{(n)} = \sum\limits_{r=1}^{k} p_{rj}^{(1)} \cdot p_{ir}^{(n-1)}, \text{ for all } n \geq 2, \end{cases}$$

are called the *Kolmogorov–Chapmann equations* associated with the homogeneous Markov chain whose transition matrix is $P = (p_{ij})_{1 \leq i,j \leq k}$ ([4]).

We denote the initial probabilities by means of the vector $q_0 = (q_0^1, \ldots, q_0^k)$, and for each $n \geq 1$ we consider the vector $q_n = (q_n^1, \ldots, q_n^k)$, where q_n^j $(1 \leq j \leq k)$ is the probability to reach the state e_j after n steps of the random process.

Notice that we have $q_n = q_0 P^n$, for each $n \geq 1$. So, in order to determine the distribution q_n it is enough to study the matrix P^n. In [2] the natural powers of the transition matrix of a finite and homogeneous Markov chain within the framework of membrane computing are computed. Moreover, the limit of the sequence $\{P^n : n \in \mathbf{N}\}$ of these matrices allows us to obtain the distribution limit in the case that it exits, and to know the stationary distribution of the process. For more details see [3] and [4].

There is a well known result [5] relating the existence of the limit of the sequence $\{P^n : n \in \mathbf{N}\}$ with the classification of the states of the Markov chain. So, we give now a classification of the states of a Markov chain and the condition by the existence of the limit.

A state e_j is *accessible* from the state e_i, denoted by $e_i \to e_j$, if there is a natural number $n > 0$ such that $p_{ij}^{(n)} > 0$. Two states e_i, e_j are *communicating states,* denoted by $e_i \leftrightarrow e_j$, if e_i is accessible from e_j and e_j is accessible from e_i. The relation of communication is an equivalence, so we can consider the equivalence classes associated with it.

The following result shows that in a finite Markov chain with k states, if we know all the paths of length $k - 1$, then we can know all the communicating states.

Proposition 1. *Let e_i, e_j be states of a finite Markov chain with k states such that e_j is accessible from e_i. Then there exists a path with length smaller than k from e_i to e_j.*

This result can be proved by substituting the nodes repeated in the path by only one copy of each of them.

A state e_i is called *recurrent* if for all e_j such that $e_i \to e_j$, then $e_j \to e_i$. On the contrary, if there exist j such that $e_i \to e_j$ but $e_j \not\to e_i$ (that is, there is a natural number m and a state e_j such that $p_{ij}^{(m)} > 0$ but $p_{ji}^{(n)} = 0$ for all $n \in \mathbf{N}$), the state e_i is called *transient.* If in an equivalence class there exists a recurrent (resp. transient) state, then every state of the class is recurrent (respectively) transient. If the class of a recurrent state e_i is formed only by this state, we say that e_i is an *absorbent state* ($p_{ij}^{(n)} = 0$ for all $e_j \neq e_i$ and for all $n \in \mathbf{N}$).

Given a state e_i such that there exists $n > 0$ and $p_{ii}^{(n)} > 0$, we define its *period* as

$$d(i) = \text{g.c.d.}\{n \geq 1 \mid p_{ii}^{(n)} > 0\}.$$

All states that belong to the same class have the same period. If the period is 1, the class is said to be *aperiodic*, otherwise we refer to it as a *periodic* class.

The problem of classification is an important one in the mathematical study of Markov chains and related stochastic processes because it allows us to study their asymptotic behavior. If we think a Markov chain as a system evolving along the time, then we are interested in analyzing how that evolution is carried out. For that, we study the existence and the uniqueness of the stationary distributions, and the convergence to stationarity starting from any initial distribution. That study is related with the number of recurrent classes of a finite Markov chain.

There are some necessary conditions for the existence of stationary distributions, that is to say, there are some results which provide us with information about the existence of the limit of the sequence of the matrix powers of a finite Markov chain ([5]).

Theorem 1. *For any Markov chain with finite states, there exists a unique stationary distribution if and only if the set of states contains precisely one recurrent class.*

Theorem 2. *A necessary and sufficient condition for the existence of a limit distribution is that there is, in the set of states of the chain, exactly one aperiodic recurrent class.*

2.2 Membrane Systems

Membrane computing is a branch of Natural Computing, considered in October 2003 by Thomson Institute for Scientific Information (**ISI**) as a *Fast Emerging Research Front* in Computer Science [9]. It was initiated at the end of 1998 by Gh. Păun (by a paper circulated at that time on web and published in 2000 [6]). Since then it has received important attention from the scientific community. Details can be found at the web page `http://psystems.disco.unimib.it`, maintained in Milano under the auspices of the European Molecular Computing Consortium, EMCC.

In short, one abstracts computing models from the structure and the functioning of living cells, as well as from the organization of cell in tissues, organs,

and other higher order structures. The main components of such a model are a cell-like *membrane structure*, in the *compartments* of which one places *multisets of symbol-objects* which evolve in a synchronous maximally parallel manner according to given *evolution rules*, also associated with the membranes. The objects can also be described by strings, they can pass through membranes, can exit the system; in turn, membranes can be divided, dissolved, created.

A large variety of computing models, called P systems, were considered in this framework, based on the fundamental concept of biological membrane; the respective models are distributed (compartmentalized) parallel computing devices, processing multisets of abstract objects by means of various types of evolution rules. Parallelism, communication, non-determinism, synchronization, dynamic architecture of the model, etc. are central concepts of the theory, with biological, mathematical, and computer science sources of inspiration.

In this way, a comprehensive and systematic interdisciplinary research area was developed, of a high generality and versatility, where models can be devised for a large range of processes where compartmentalization and multiset processing are natural ingredients. Thus, although the initial goal of membrane computing was only to learn new ideas, tools, techniques from cell biology to the help of standard computers, much in the same way as, e.g., evolutionary computing suggests algorithms to be implemented on the electronic computer, the membrane computing became a new framework for building models for a large variety of processes, especially from biology (cell biology, tissues, populations of bacteria, controlling networks of complex phenomena, tumor growth, etc.), but also from linguistics, management, with several applications to computer science (computer graphics, approximative solutions to computationally hard problems, modeling parallel architectures, cryptography).

Most of these models were proven to be computationally universal, able to compute whatever a Turing machine can compute. In the case when an enhanced parallelism is available, by means of membrane division, string-object replication, or membrane creation, polynomial (often linear) time solutions to **NP**-complete problems were found.

In many variants, P systems are seen as devices of a *generative* nature, that is, from a given initial configuration several distinct computations may be developed, in a non–deterministic manner, producing different outputs.

In this paper we work with P systems with external output and performing *computing* tasks. For example, if a certain natural number, n, is encoded by the multiplicity of a special object in the initial configuration and we consider the cardinality of the multiset contained in the environment of a halting configuration as the result of a successful computation, then we can interpret that to mean that the system *computes* a partial function from natural numbers onto sets of natural numbers.

In the following, we assume that the reader is familiar with the basic notions of P systems, and we refer, for details, to [7].

3 Computing the Classification of the Steps of a Finite Markov Chain

3.1 Designing a P System

The goal of this paper is to obtain the classification of the states of a finite and homogeneous Markov chain within the framework of the cellular computing with membranes.

Let $P_k = (p_{ij})_{1\leq i,j\leq k}$ be a boolean matrix associated with a finite and homogeneous Markov chain of order k such that $p_{ij} = 0$ if the transition from e_i to e_j is not possible, and $p_{ij} = 1$ if the transition from e_i to e_j is possible; that is, P_k is the adjacency matrix of the directed graph associated with the Markov chain.

The solution presented in this paper is a *semi–uniform* solution to the problem of classification, in the following sense: we give a family $\mathbf{\Pi} = \{\Pi(P_k) : k \in \mathbf{N}\}$, associating with P_k a P system with external output, such that:

- There exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(P_k)$ from P_k.
- The output of the P system $\Pi(P_k)$ provides the classification of the k states of the Markov chain as well as the period of the states.

We associate with the matrix P_k a P system of degree 4 with external output,

$$\Pi(P_k) = (\Gamma(P_k), \mu(P_k), \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \mathcal{M}_4, R, \rho)$$

defined as follows:

- Working alphabet:

$$\begin{aligned}\Gamma(P_k) = &\{a_{ij},\ b_{ij},\ d_{ij},\ t_{ij} :\ 1 \leq i,j \leq k,\} \cup \{c_r :\ 0 \leq r \leq 2k+2\} \cup \\ &\{t_{ijur} :\ 1 \leq i,j,u \leq k,\ 0 \leq r \leq k\} \cup \{\beta_i :\ 0 \leq i \leq \alpha+1\} \cup \\ &\{s_{ijr} :\ 1 \leq i,j \leq k,\ 0 \leq r \leq k\} \cup \{A_{i1},\ \gamma_i :\ 1 \leq i \leq k\} \cup \\ &\{T_{ij},\ R_{ij} :\ 1 \leq i,j \leq k\}\end{aligned}$$

  where $\alpha = 2k + 4 + \lceil \lg_2 k \rceil + \frac{(k-1)(k+2)}{2}$.
- Membrane structure: $\mu(P_k) = [_1\ [_2\ [_3\ [_4\ ]_4\ ]_3\ ]_2\ ]_1$.
- Initial multisets:

$$\mathcal{M}_1 = \emptyset;\ \mathcal{M}_2 = \{\beta_0\};\ \mathcal{M}_3 = \{c_0\};$$
$$\mathcal{M}_4 = \{s_{ii0} \quad t_{ij}^{p_{ij}(k-1)} :\ 1 \leq i,j \leq k\}.$$

- The set R of evolution rules consists of the following rules:
  - Rules in the skin membrane labeled by 1:

$$r_1 = \{b_{ij}b_{ji} \to a_{ij}a_{ji} :\ 1 \leq i < j \leq k\}$$
$$r_2 = \{b_{ij} \to \gamma_i;\quad \gamma_i a_{ij} d_{ip} d_{jp} \to (T_{ip}T_{jp}, out) :\ 1 \leq i,j,p \leq k\}$$
$$r_3 = \{\gamma_i d_{ip} \to (T_{ip}, out) :\ 1 \leq i,j,p \leq k\}$$
$$r_4 = \{a_{ij}d_{ip} \to (R_{ip}, out) :\ 1 \leq i,j,p \leq k\}$$
$$r_5 = \{d_{i1} \to (A_{i1}, out) :\ 1 \leq i \leq k\}$$

- Rules in the membrane labeled by 2:
  $r_6 = \{b_{ij}^2 \to b_{ij} : \ 1 \leq i, j \leq k\} \cup \{\beta_i \to \beta_{i+1} : \ 0 \leq i \leq \alpha\} \cup \{\beta_{\alpha+1} \to \delta\}.$
  $r_7 = \{d_{ij}^2 \to d_{ij} : \ 1 \leq i, j \leq k\}$
  $r_8 = \{d_{ij} d_{i(j+l)} \to d_{ij} d_{il} : \ 1 \leq i \leq k, \ 2 \leq j + l \leq k\}$

- Rules in the membrane labeled by 3:
  $r_9 = \{t_{ijur} \to (t_{ij} s_{uj(r+1)}, in_4) \ b_{uj} : \ p_{ij} = 1, u \neq j, 1 \leq i, j, u \leq k, 0 \leq r < k\}$
  $r_{10} = \{t_{ijuk} \to (t_{ij}, in_4) \ b_{uj} : \ p_{ij} = 1, u \neq j, 1 \leq i, j, u \leq k\}$
  $r_{11} = \{t_{ijjr} \to (t_{ij}, in_4) \ d_{j(r+1)} : \ p_{ij} = 1, 1 \leq i, j \leq k, 0 \leq r < k\}$
  $r_{12} = \{t_{ijjk} \to (t_{ij}, in_4) : \ p_{ij} = 1, 1 \leq i, j \leq k\}$
  $r_{13} = \{c_r \to c_{r+1} : \ 0 \leq r \leq 2k + 1\} \ \cup \{c_{2k+2} \to \delta\}$
- Rules in the membrane labeled by 4:
  $r_{14} = \{s_{uir} t_{i1}^{p_{i1}} \ldots t_{ik}^{pik} \to (t_{i1ur}^{p_{i1}} \ldots t_{ikur}^{pik}, out) : \ 1 \leq u, i \leq k, \ 0 \leq r \leq k\}.$

– The partial order relation ρ over R consists of the following relations on the rules of R:
  - Priority relation in the skin membrane: $\{r_1 > r_2 > r_3 > r_4 > r_5\}$.
  - Priority relation in the membrane labeled by 2: $\{r_7 > r_8\}$.
  - Priority relation in the membranes labeled by 3: $\emptyset$.
  - Priority relation in the membranes 4: $\emptyset$.

Remark 1. Let us observe that the resources initially required for constructing the P system P_k are the following:

– Size of the alphabet: $\theta(k^4)$.
– Intitial number of membranes: 4.
– Number of rules: $O(k^4)$.
– Maximal length of a rule: $O(k)$.
– Number of priority relations: $O(k^6)$.

3.2 An Overview of Computations

At the beginning, the skin membrane is empty. The membrane labeled by 2 only contains the object β_0 which is a counter used to dissolve that membrane in the $(\alpha+2)$–th step, where $\alpha = 2k+4+\lceil lg_2 k \rceil + (k-1)(k+2)/2$. The membrane labeled by 3 contains the object c_0 which is a counter used to dissolve the membrane 2 in the $(2k + 3)$-th step. Initially, the membrane labeled by 4 contains: (a) objects s_{ii0} $(1 \leq i \leq k)$ encoding the states e_i of the chain; and (b) objects t_{ij} $(1 \leq i, j \leq k)$ encoding the elements p_{ij} of the boolean matrix associated to the transition matrix of the Markov chain.

In the first $2k + 3$ steps one applies rules only in the internal membranes labeled by 2, 3, and 4. During this (so called) first stage, we determine the accessibility between states (encoded by the objects b_{ij} meaning that we can

reach e_j from e_i) as well as the recurrent time of each state (encoded by the objects d_{ij} meaning that there exists a path from e_i to e_i with length j). In the even steps, the rules of membrane 4 will consume all the objects s_{uir} and some objects t_{ij}, sending to membrane 3 some objects t_{ijur}. In the odd steps, only rules in membrane 3 are applied (but not in membrane 4, because there do not exist objects s_{uir} in that membrane), sending new objects t_{ij} and objects $s_{uj(r+1)}$ (with $u \neq j$) to that membrane and producing objects b_{uj} and d_{jr} in membrane 3. The first stage finalizes in the configuration C_{2k+3} when the rule $c_{2k+3} \to \delta$ dissolves membrane 3. In this moment we have some objects t_{ij} (with $1 \leq i, j \leq k$) in membrane 4, objects d_{jr}, b_{uj} (with $1 \leq j, u, r \leq k$) and the object β_{2k+3} in membrane 2 (notice that in each step of this first stage the rule $\beta_i \to \beta_{i+1}$ of membrane 2 has been carried out). The skin region is empty.

The second stage begins with the execution of the $(2k+4)$–th step. During this stage we eliminate repeated copies of objects b_{ij} and d_{ij} in membrane 2, and we compute the period of each state (encoded in the second subscript of the objects d). The rules of membrane 2 permit transforming two copies of the object b_{ij} and d_{ij} into one copy, and the period of each state e_i is calculated by means of the rules of type r_8. For that, we need at most $\alpha = 2k+4+\lceil lg_2 k \rceil + (k-1)(k+2)/2$. steps. This stage finalizes when the rule $\beta_{\alpha+1} \to \delta$ dissolves membrane 2 in the $(\alpha+2)$–th step. This stage is a non-deterministic one.

Finally, the third stage is the output phase, and begins with the execution of the $(\alpha+3)$–step. In this stage the objects $b_{ij}b_{ji}$ are transformed into the objects $a_{ij}a_{ji}$ by means of the rule r_1 (meaning that the states e_i and e_j belongs to the same equivalence class). When this rule cannot be applied, then the transient objects are expelled to the environment applying the rules of types r_2 and r_3. After that, the rule r_4 sends the recurrent states and their period to the external environment. The process finalized when the rule r_5 sends the absorbent states.

3.3 Formal Verification

Given a computation $\mathcal{C}$ of the P system $\Pi(P_k)$, for each $m \in \mathbf{N}$ we denote by $\mathcal{C}_m$ the configuration of the system obtained after the execution of m steps. For each label $l \in \{1, 2, 3, 4\}$, we denote by $C_m(l)$ the multiset of objects contained in the membrane labeled by l in the configuration $\mathcal{C}_m$. Also, we denote by $C_m(env)$ the content of the environment of the system in the configuration $\mathcal{C}_m$.

First of all, we show that during the first stage the objects s_{ijr} codify the existence of a path from e_i to e_j with length r, and the objects t_{ijur} codify the existence of a path from e_u to e_j with length r and with e_i next to last node.

Lemma 1. *For each r such that $1 \leq r \leq k$ we have the following:*

(a) *If $r = 1$, then for each i, j such that $1 \leq i, j \leq k$, the object t_{iji0} belongs to $C_1(3)$ if and only if there exists a path from e_i to e_j with length 1 and with e_i being next to last node.*
If $r > 1$, then for each i, j, u such that $1 \leq i, j, u \leq k$, the object $t_{iju(r-1)}$ belongs to $C_{(2r-1)}(3)$ if and only if there exists a path from e_u to e_j with length r and with e_i being next to last node.

(*b*) *For each* i, j *such that* $1 \leq i, j \leq k$, $i \neq j$, *the object* s_{ijr} *belongs to* $C_{2r}(4)$ *if and only if there exists a path from* e_i *to* e_j *with length* r.

Proof. We prove the lemma by induction on r.

- Let us suppose that $r = 1$.
  (*a*) Let i, j be such that $1 \leq i, j \leq k$.
  If $t_{iju0} \in C_1(3)$, having in mind the composition of the initial configuration, there exists objects s_{ii0} and t_{ij} in $C_0(4)$. So, $p_{ij} = 1$ and (e_i, e_j) is an arc of the graph associated with the Markov chain. Hence, there exists a path from e_i to e_j with length 1 and with e_i next to last node. Conversely, if there exists a path from e_i to e_j with length 1 and with e_i next to last node, then $p_{ij} = 1$. So, the object t_{ij} belongs to $C_0(4)$. Having in mind that $s_{ii0} \in C_0(4)$, and applying the rules of type r_{14} we have $t_{iji0} \in C_1(3)$.
  (*b*) Let i, j be such that $1 \leq i, j \leq k$, $i \neq j$.
  Let us suppose that $s_{ij1} \in C_2(4)$. Then that object has been produced by an object $t_{iji o}$ belongs to $C_1(3)$ and applying the rules of type r_9. From *a*) we deduce that there exists a path from e_i to e_j with length 1 (and with e_i next to last node).
  If there exists a path from e_i to e_j with length 1, then from *a*) we deduce that the object t_{iji0} belongs to $C_1(3)$. Applying the rule of type r_9 we obtain that $s_{ij1} \in C_2(4)$.
- Let $r \geq 1$ and $r < k$ and let us suppose that conditions (*a*) and (*b*) hold for r. Let us show that these conditions hold for $r + 1$.
  (*a*) Let i, j, u be such that $1 \leq i, j, u \leq k$.
  If the object t_{ijur} belongs to $C_{2r+1}(3)$, then in the $(2r + 1)$–th step the rules of type r_{14} has been applied in membrane 4, in order to produce the object t_{ijur}. Then, the objects s_{uir} and t_{ij} must belongs to $C_{2r}(4)$. By the induction hypothesis there exists a path from e_u to e_i of length r. Having in mind that $t_{ij} \in C_{2r}(4)$, it follows that (e_i, e_j) is an arc of the graph associated. Consequently there exists a path from e_u to e_j with length $r + 1$ with e_i next to last node.
  Let us suppose that there exists a path from e_u to e_j with length $r+1$ with e_i next to last node. Then there is a path from e_u to e_i of length r. By the induction hypothesis, the object s_{uir} belongs to $C_{2r}(4)$. Moreover, $p_{ij} = 1$ because (e_i, e_j) is an arc of the graph associated, so t_{ij} belongs to $C_{2r}(4)$. Applying the rules of type r_{14}, we have $t_{ijur} \in C_{2r+1}(3)$.
  (*b*) Let i, j be such that $1 \leq i, j \leq k$, $i \neq j$.
  If the object $s_{ij(r+1)}$ belongs to $C_{2r+2}(4)$, then there exists u $(1 \leq u \leq k)$ such that the object t_{ujir} belongs to $C_{2r+1}(3)$. By the induction hypothesis, there exists a path form e_i to e_j of length $r + 1$ with e_i next to last node. Then, there exists a path from e_i to e_j of length $r + 1$.
  Conversely, let us suppose that there exists a path from e_i to e_j of length $r + 1$. Let u be such that e_u is the next to last node of this path. By induction hypothesis, we have $t_{ujir} \in C_{2r+1}(3)$. Applying the rules of type r_9 we obtain that the object $s_{ij(r+1)}$ belongs to $C_{2r+2}(4)$. □

Lemma 2. *For each r such that $1 \leq r \leq k$ we have the following:*

(a) There are i, j, u such that $1 \leq i, j, u \leq k$, $t_{iju(r-1)} \in C_{2r-1}(3)$, $s_{ijr} \in C_{2r}(4)$.
(b) For all i, j, u, z such that $1 \leq i, j, u, z \leq k$, we have:

$$t_{ijuz} \notin C_{2r}(3),\ s_{ijz} \notin C_{2r-1}(4), t_{ij}^{p_{ij}(k-1)} \in C_{2r}(4)$$

Proof. By induction on r. First of all, recall that

$$C_0(4) = \{s_{ii0} t_{ij}^{p_{ij}(k-1)} :\ 1 \leq i, j \leq k\},\ C_0(3) = \{c_0\}.$$

Let i, j be such that $1 \leq i, j \leq k$ and $p_{ij} = 1$. Applying the rules of type r_{14} at the initial configuration we have $t_{iji0} \in C_1(3)$. Then, applying the rules of type r_9 in the second step we have $s_{ij1} \in C_2(4)$. Moreover, each object t_{ij} that has evolved in the first step, returns to membrane 4 in the next step. So, $t_{ij}^{p_{ij}(k-1)} \in C_2(4)$, for all i, j $(1 \leq i, j \leq k)$.

Having in mind that in the first step all objects s_{ii0} are consumed, we have $s_{ijz} \notin C_1(4)$, for all i, j, z $(1 \leq i, j, z \leq k)$ Hence, $t_{ijuz} \notin C_2(3)$, for all i, j, u, z $(1 \leq i, j, u, z \leq k)$.

Assuming the result holds for $r < k$ $(r \geq 1)$, we prove the result holds for $r+1$.

By the induction hypothesis, there exist i, j $(1 \leq i, j \leq k)$ such that $s_{ijr} \in C_{2r}(4)$. But there is u $(1 \leq u \leq k)$ such that $t_{uj} \in C_{2r}(4)$; applying the rules of type r_{14} we have we have $t_{ijur} \in C_{2r+1}(3)$. Then, applying the rules of type r_9 in the next step we have $s_{uj(r+1)} \in C_{2r+2}(4)$. Moreover, each object t_{ij} that has evolved in the r–th step, returns to membrane 4 in the next step. So, $t_{ij}^{p_{ij}(k-1)} \in C_{2r+2}(4)$, for all i, j $(1 \leq i, j \leq k)$.

Having in mind that in the r–th step all objects s_{ijr} which belong to $C_{2r}(4)$ have evolved, we have $s_{ijz} \notin C_{2r+1}(4)$, for all i, j, z $(1 \leq i, j, z \leq k)$ Hence, $t_{ijuz} \notin C_{2r+2}(3)$, for all i, j, u, z $(1 \leq i, j, u, z \leq k)$. □

Proposition 2. *For each i, j such that $1 \leq i, j \leq k$ we have the following:*

(1) If $i \neq j$, then the following assertions are equivalent:
  (a) There exists a path from e_i to e_j.
  (b) The object b_{ij} belongs to $C_{2k+2}(3)$.
  (c) The object b_{ij} belongs to $C_{2k+3}(2)$.
(2) The following conditions are equivalent
  (a) There exists a path from e_i to e_i with length j.
  (b) The object d_{ij} belongs to $C_{2k+2}(3)$.
  (c) The object d_{ij} belongs to $C_{2k+3}(2)$.

Proof. Let i, j be such that $1 \leq i, j \leq k$.

(1) Let $i \neq j$ and let us suppose that there exists a path from e_i to e_j. Let $r \geq 1$ be the length of that path r, and let e_u be the next to last node of that path. From Lemma 1, we have $t_{uji(r-1)} \in C_{2r-1}(3)$. Applying the rules of type r_9 or r_{10} we obtain that $b_{ij} \in C_{2r}(3)$. Hence $b_{ij} \in C_{2k+2}(3)$.

Conversely, let us suppose that $b_{ij} \in C_{2k+2}(3)$. Then, from Lemma 2 there exists r $(1 \geq r \leq k)$ such that $t_{uji(r-1)} \in C_{2r-1}(3)$. From Lemma 1 we deduce that there exists a path from e_i to e_j.
Obviously, $b_{ij} \in C_{2k+2}(3) \Longleftrightarrow b_{ij} \in C_{2k+3}(2)$.

(2) Let us suppose that there exists a path from e_i to e_i of length j. Then, there exists a state e_u and a path from e_i to e_u of length $j-1$, and with (e_u, e_i) being an arc of the associated graph. From Lemma 1, the object $t_{uii(j-1)}$ belongs to $C_{2j-1}(3)$. Applying the rules of type r_{11} or r_{12} we have $d_{ij} \in C_{2j}(3)$. Hence, $d_{ij} \in C_{2k+2}(3)$.

Conversely, let us suppose that $d_{ij} \in C_{2k+2}(3)$. Then, from Lemma 2 there exists r $(1 \geq r \leq k)$ such that $t_{uii(j-1)} \in C_{2r-1}(3)$. From Lemma 1 we deduce that there exists a path from e_i to e_i with length j.

Obviously, $d_{ij} \in C_{2k+2}(3)$ if and only if $d_{ij} \in C_{2k+3}(2)$. □

Proposition 3. *If* $\alpha = 2k + 4 + \lceil lg_2 k \rceil + (k-1)(k+2)/2$, *then:*

$$C_{\alpha+1}(2) = \{b_{ij} : 1 \leq i, j \leq k, i \neq j, \text{ there is a path from } e_i \text{ to } e_j\} \cup \{d_{ip} : 1 \leq i, p \leq k, \ p \text{ is the period of the state } e_i\} \cup \{\beta_{\alpha+1}\}.$$

$$C_{\alpha+2}(1) = \{b_{ij} : 1 \leq i, j \leq k, i \neq j, \text{ there is a path from } e_i \text{ to } e_j\} \cup \{d_{ip} : 1 \leq i, p \leq k, \ p \text{ is the period of the state } e_i\}.$$

Proof. Applying repeatedly the rules $\beta_i \longrightarrow \beta_{i+1}$ $(0 \leq i \leq \alpha)$ starting from the initial configuration, we have $\beta_{\alpha+1} \in C_{\alpha+1}(2)$.

From Proposition 2 we deduce that in membrane 2 of the configuration C_{2k+3} we have objects b_{ij}, with different multiplicities, such that there is a path from the state e_i to state e_j, and objects d_{ij}, with multiplicity 1, such that there is a path from the state e_i to state e_i with length j. Then, applying the rules of type r_6 in, at most, $\lceil \lg_2 k \rceil$ steps, we get that the multiplicity of each object is 1. Simultaneously, applying the rules of type r_7 and (8) in at most $\lceil \lg_2 k \rceil + (k-1)(k+2)/2$ steps we produce the objects d_{ip}, where p is the greatest common divisor of $\{d_{ij} : d_{ij} \in C_{2k+3}(2)\}$.

In the step $\alpha + 2$, membrane 2 is dissolved by executing the rule $\beta_{\alpha+1} \to \delta$. □

Theorem 3. *Let* C_f *be the final configuration of the computation* $\mathcal{C}$ *of the system* $\Pi(P_k)$. *Then:*

(*a*) *The state* e_i *is transient with period* p *if and only if* $T_{ip} \in C_f(env)$.
(*b*) *The state* e_i *is recurrent (and not absorbent) with period* p *if and only if* $R_{ip} \in C_f(env)$.
(*c*) *The state* e_i *is absorbent (with period* 1*) if and only if* $A_{i1} \in C_f(env)$.

Proof. (a) Let us suppose that e_i is a transient state. If the equivalence class of e_i has more than one element, then we can apply the rules of type r_1 in membrane 1 of the configuration $C_{\alpha+2}$ producing objects a_{ij} and a_{ji}. In this case, there exists r $(1 \leq r \leq k)$ such that the object b_{ir} belongs to $C_{\alpha+2}(1)$ but $b_{ri} \notin C_{\alpha+2}(1)$. So, in the $(\alpha+4)$–th step the object γ_i is produced applying the rules of type r_2. Then in the next step (and using the object d_{ip}) we obtain that $T_{ip} \in C_{\alpha+5}(env)$, where p is the period of e_i (from Proposition 3).

If the equivalence class of e_i is a singleton, then the rules of type r_1 cannot be applied in the configuration $C_{\alpha+2}$. So, we apply the rules of type r_2 producing

the object γ_i that in the next step produces (together with the object d_{ip}) the object T_{ip} in the environment (that is, $T_{ip} \in C_{\alpha+4}(env)$).

Reciprocally, let us suppose that $T_{ip} \in C_{\alpha+4}(env)$). Then, the object γ_i must be generated in order to can apply the rules of types r_2 and/or r_3. If only the rules of type r_2 are applied, then there are j, j' $(1 \leq j, j' \leq k)$ such that e_j is accessible from e_i and e_i is accessible from e_j, and e_i, $e_{j'}$ are communicating states. Hence, e_i is a transient state whose equivalence class has more than one object. If the rules of type r_3 are applied, then e_i is a transient state whose equivalence class is a singleton.

(b) Let us suppose that the state e_i is recurrent (and not absorbent) with period p. Then, the equivalence class of e_i has more than one object and there is no transient state belongs to that class. So, the rules of type r_1 will be applied in the configuration $C_{\alpha+2}$ and the object γ_i cannot be produced. Hence, applying the rules of type r_4 in the next configuration we have $R_{ip} \in C_{\alpha+4}(env)$.

Reciprocally, if $R_{ip} \in C_f(env)$ then the rule $a_{ij}d_{ip} \rightarrow (R_{ip}, out)$ has been applied (for some j, p with $1 \leq j, p \leq k$). For that, the objects a_{ij} and a_{ji} have been produced and the object γ_i has not been generated. Consequently, the state e_i is recurrent and its equivalence class has more than one object (that is, it is not an absorbent state).

(c) Let us suppose that the object e_i is absorbent (consequently its period is 1). In this case, it equivalence class is a singleton. So, there is no j $(1 \leq j \leq k)$ such that b_{ij} and b_{ji} belongs to $C_{\alpha+2}(1)$. Then the rules of type r_1 are not applicable for i. Applying the rule $d_{i1} \rightarrow (A_{i1}, out)$ we obtain that $A_{i1} \in C_{\alpha+3}(env)$.

Reciprocally, if $A_{i1} \in C_f(env)$, then the object d_{i1} belongs to membrane 1 in the next to last configuration. So, the objects a_{ij} has not been produced. Then, the state is recurrent and its equivalence class has only one object. □

Remark 2. Let us note that the number of steps of the computation of the P system P_k is either $\alpha + 3$ or $\alpha + 4$. That is, the number of steps is quadratic in the number of states of the Markov chain.

4 Conclusions

One of the central issues in Markov chain theory is the asymptotic long–term behavior of Markov chains.

Due to different results concerning the existence (and the uniqueness) of a stationary distribution, the problem of classification of states is an important one in the mathematical study of Markov chains and related stochastic processes.

In this paper we give an efficient (*semi–uniform*) solution of the problem of classification in the framework of the cellular computing with membranes. The solution is semi–uniform because for each adjacency matrix of the directed graph associated with a Markov chain, a specific P system with external output is designed. The solution is efficient, because it is quadratic in the number of states of the Markov chain. Furthermore, the amount of resources initially required to construct the system is polynomial in the order of the Markov chain.

The paper also provides a new example of formal verification of P systems designed to solve a problem (in this case a problem of classification, not a decision problem), following a specific methodology. These examples are always interesting, for instance, in order to find systematic processes of formal verification in a model of computation oriented to machines, like the cellular model, a case where it is well known that the mechanisms of verification are often a very hard task.

Acknowledgement

The third author wishes to acknowledge the support of the project TIN2005-09345-C04-01 of the Ministerio de Educación y Ciencia of Spain, co–financed by FEDER funds, and of the Project of Excelence TIC 581 of the Junta de Andalucia.

References

1. M. Cardona, M.A. Colomer, J. Miró, A. Zaragoza, A step towards DNA computation model. Submitted, 2006.
2. M. Cardona, M.A. Colomer, M.J. Pérez–Jiménez, A. Zaragoza, Handling Markov chains with membrane computing. *Lecture Notes in Computer Science*, 4135 (2006), 72–85.
3. O. Häggström, *Finite Markov Chains and Algorithmic Applications*. London Mathematical Society, Cambridge University Press, 2003.
4. R. Nelson, *Probability, Stochastic Processes, and Queueing Theory: The Mathematics of Computer Performance Modeling*. Springer-Verlag, New York, 1995.
5. A.N. Shiryayev. *Probability*. GTM 95, Springer, 1984.
6. Gh. Păun, Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for Computer Science-TUCS Report* Nr. 208, 1998.
7. Gh. Păun, *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
8. Gh. Păun, G. Rozenberg, A guide to membrane computing. *Theoretical Computer Science*, 287 (2002), 73–100.
9. ISI web page: `http://esi-topics.com/erf/october2003.html`

Partial Knowledge in Membrane Systems: A Logical Approach

Matteo Cavaliere[1] and Radu Mardare[2]

[1] Microsoft Research - University of Trento
Centre for Computational and Systems Biology, Trento, Italy
matteo.cavaliere@msr-unitn.unitn.it
[2] D.I.T, University of Trento
Trento, Italy
mardare@dit.unitn.it

Abstract. We propose a logic for specifying and proving properties of membrane systems. The main idea is to approach a membrane system by using the "point of view" of an external observer. Observers (as epistemic agents) accumulate their knowledge from the partial information they collect by observing subparts of the system and by applying logical reasoning to this information. We provide a formal framework to combine and interpret distributed knowledge in order to recover the complete knowledge about a membrane system. The proposed logic can be used to model biological situations where information concerning parts of the biological system is missing or incomplete.

1 Motivations

Abstracted as a multi-agent system, a biological system reflects interactive, concurrent, and distributed behaviors and, in general, the complex evolutions of biological systems. The success in dealing with this complexity depends on the mathematical model chosen as abstraction of the system.

Consider, for example, the *immune system* [1]. This is constituted by a network of cells, tissues, and organs that work together to defend the body against attacks by foreign invaders – microbes, germs, bacteria, viruses, parasites, etc. The immune system's job is to keep them out or, failing that, to seek out and destroy them. The immune system functions due to an elaborate and dynamic communications network. Millions of cells, organized into sets and subsets, gather in clouds swarming around a hive and pass information back and forth.

Suppose now that we are interested in modeling the interaction of our body with a given virus. Excepting the immune system, our body contains also other subsystems, but we can decide that, for the given situation, all the other parts are meaningless. So we decide to ignore them. For instance, if we consider the case in which the virus is already present in our body, the first approximation of the biological reality will consider a main system (our body) in which are present two subsystems – the virus and the immune system. Going deeper, the

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 279–297, 2006.

first interaction between the two subsystems involves the *innate immune system*, which is just a subsystem of the immune system comprised of hereditary components that provide an immediate "first-line" of defense to continuously ward off pathogens. This subsystem is able to annihilate "well-known" viruses. If this is the real situation, then modeling only the innate immune system in relation with the virus is enough for comprehending the biological phenomenon. But if the virus is unknown, then we might need to go deeper with modeling and, in addition to the innate immune system, to model also *phagocytic cells*. These are cells that represents the "second-line" of defence for our body. They can analyze unknown entities, destroy viruses and learn the structures of the destroyed entities. In particular, the immune system is able to design special cells for fighting with peculiar types of viruses. Hence, on this level, the modeling have to be more specific representing also other subsystems of the immune system.

Depending on the complexity of the biological properties we want to consider, we can go as deep as necessary with representing the biological entities involved. More complex models provide more accurate information. Still, as the costs of modeling and simulation grows with complexity of the model, we have to find the right level of abstraction that gives, with acceptable costs, the information we are looking for. Observe that in biology, as in all the empirical sciences, *we cannot hope to reach the level of having complete information concerning a biological phenomenon.* Thus, no matter how complex is the model we choose, there exists always properties requiring a bigger complexity.

In other words, *we always work with partial (observed) knowledge* about biological systems and based on this incomplete information we model or simulate biological phenomena. In this paper we show how it is possible to manage incomplete information concerning membrane systems. The work done here can be seen as related to [7] where a formal observer has been introduced to investigate the formal behavior of a membrane system. However in [7] the observer was mainly used to extend the computing power of the observed device. In this paper, the observer is an epistemic agent able to compute knowledge and is used to analyze situations in which the knowledge about the observed system is partial, incomplete or missing.

2 Formal Language Preliminaries

Membrane systems are based on *formal language theory* and *multiset rewriting*. We now briefly recall the basic theoretical notions used in this paper. For more details the reader can consult standard books, such as [8] and the corresponding chapters of the handbook [17].

Given the set A we denote by $|A|$ its cardinality and by $\emptyset$ the empty set. We denote by $\mathbb{N}$ and by $\mathbb{R}$ the set of natural and real numbers, respectively.

As usual, an *alphabet* V is a finite set of symbols. By V^* we denote the set of all strings over V. By V^+ we denote the set of all strings over V excluding

the empty string. The empty string is denoted by λ. The *length* of a string v is denoted by $|v|$. The concatenation of two strings $u, v \in V^*$ is written uv.

A *multiset* is a set where each element may have a multiplicity. Formally, a multiset over a set V is a map $M : V \to \mathbb{N}$, where $M(a)$ denotes the multiplicity of the symbol $a \in V$ in the multiset M.

For multisets M and M' over V, we say that M is *included in* M' if $M(a) \leq M'(a)$ for all $a \in V$. Every multiset includes the *empty multiset*, defined as M where $M(a) = 0$ for all $a \in V$.

The *sum* of multisets M and M' over V is written as the multiset $(M + M')$, defined by $(M + M')(a) = M(a) + M'(a)$ for all $a \in V$. The *difference* between M and M' is written as $(M - M')$ and defined by $(M - M')(a) = max\{0, M(a) - M'(a)\}$ for all $a \in V$. We also say that $(M + M')$ is obtained by *adding* M to M' (or viceversa) while $(M - M')$ is obtained by *removing* M' from M. For example, given the multisets $M = \{a, b, b, b\}$ and $M' = \{b, b\}$, we can say that M' is included in M, that $(M+M') = \{a, b, b, b, b, b\}$ and that $(M-M') = \{a, b\}$.

A multiset M can be expressed in the forms $(a, M(a))$ or $a^{M(a)}$, for all $a \in V$. If the set V is finite, e.g. $V = \{a_1, \ldots, a_n\}$, then the multiset M can be explicitly described as $\{(a_1, M(a_1)), (a_2, M(a_2)), \ldots, (a_n, M(a_n))\}$. The *support* of a multiset M is defined as the set $supp(M) = \{a \in V \mid M(a) > 0\}$. A multiset is empty (hence finite) when its support is empty (also finite).

A compact notation can be used for finite multisets: if $M = \{(a_1, M(a_1)), (a_2, M(a_2)), \ldots, (a_n, M(a_n))\}$ is a multiset of finite support, then the string $w = a_1^{M(a_1)} a_2^{M(a_2)} \ldots a_n^{M(a_n)}$ (and all its permutations) precisely identify the symbols in M and their multiplicities. Hence, given a string $w \in V^*$, we can say that it identifies a finite multiset over V, written as $M(w)$, where $M(w) = \{a \in V \mid (a, |w|_a)\}$. For instance, the string bab represents the multiset $M(w) = \{(a, 1), (b, 2)\}$, that is the multiset $\{a, b, b\}$. The empty multiset is represented by the empty string λ.

3 Membrane Systems with Symbol-Objects

We recall the basic notions of membrane systems (also called P systems) with symbol-objects. The reader can find further details in the monograph [16]. An updated bibliography of the field can be found at the P systems web-page [18].

Definition 1 (Membrane system with symbol-objects). *Given a finite set of objects O and an infinite set of labels Lab, we consider the following family of constructs*

$$\mathbb{P} = \{(\mu, w_{j_1}, w_{j_2}, \cdots, w_{j_m}, R_{j_1}, R_{j_2}, \cdots, R_{j_m}) \mid j_i \in Lab, \text{ for } i = 1..m\}.$$

where

- *μ is a* membrane structure *consisting of m membranes arranged in an hierarchical structure enclosed in a main membrane, called* skin membrane*. The skin membrane separates the system from the surrounding* environment*; the*

membranes (and hence the regions that they delimit/enclose) are injectively labeled over $Lab_{\Pi} = \{j_1, j_2, \cdots, j_m\} \subset Lab$*; we* convey to label by j_1 the skin membrane.

- $w_{j_1}, w_{j_2}, \cdots, w_{j_m}$ *are strings that represents multisets over* O *associated with regions* $j_1, j_2, \cdots, j_m$*, respectively.*
- $R_{j_1}, R_{j_2}, \cdots, R_{j_m}$ *are finite sets of* evolution rules *over* O*, associated to regions* $j_1, j_2, \cdots, j_m$*, respectively. An evolution rule is of the form* $u \to v$*, where* u *is a string over* O *and* v *is a string over* $\{a_{here}, a_{out} \mid a \in O\} \cup \{a_{in_I} \mid a \in O, I \subseteq Lab\}$*. The symbols* here, out, in_I with $I \subseteq Lab$ *are called* target indications*. To simplify the notation the target indication* here *is omitted.*

An element $\Pi = (\mu, w_{j_1}, w_{j_2}, \cdots, w_{j_m}, R_{j_1}, R_{j_2}, \cdots, R_{j_m}) \in \mathbb{P}$ is called *membrane system with symbol-objects*, of degree m. We denote by 0 the membrane system of degree 0. We call *atomic membrane system* a system a system of degree 1 having either the set of rules empty or the multiset of objects empty; if its unique membrane (which is also the skin membrane) is labelled by i and $R_i = \emptyset$ while its multiset is $w_i \in O^*$, then we denote it by $[w_i]_i$; if $w_i = \lambda$ and $R_i \neq \emptyset$ then we denote it by $[R_i]_i$; if $R_i = \emptyset$ and $w_i = \lambda$ we convey to denote it by $[0]_i$.

Given a membrane system Π, an *evolution of* Π is a *sequence of membrane systems* $\langle \Pi_0, \Pi_1, \Pi_2, \cdots \rangle$ where $\Pi_0 = \Pi$ and, for $i \geq 0$, each Π_{i+1} is obtained by applying once one of the associated evolution rules in one of the regions of Π_i. Rule and region are chosen in a non-deterministic manner. The remainder of the system Π_i (objects not involved in the application of the rule, set of rules, membrane structure, labeling of the membranes) is left unchanged in Π_{i+1}.

The passage from Π_i to Π_{i+1} using the rule r in region j of Π_i is called *transition* and is denoted by $\Pi_i \xrightarrow{r_j} \Pi_{i+1}$. [1]

The *application of an evolution rule* $r : u \to v \in R_j$ in the region $j \in Lab_{\Pi}$ means to remove the multiset of objects identified by u from region j, and to add the objects specified by the multiset v, in the regions specified by the target indications associated to each object in v. In particular, if v contains an object a with target indication *here*, then the object a will be placed in the region j where the evolution rule has been applied. If v contains an object a with target indication *out*, then the object a will be moved to the region immediately outside the region j (this can be the environment if the region where the rule has been applied is the *skin* membrane). If v contains an object a with target indication in_I, with $I \subseteq Lab$, then the object a is moved from the region j and placed in a non-deterministic way into a region $i \in I$ (this can be done only if such region $i \in I$ is immediately inside region j; otherwise the evolution rule $u \to v$ cannot be applied).

We call *contents of membrane* j *in* Π, the multiset of objects and the membranes (together with their contents) contained in region j of Π.

[1] The reader familiar with membrane systems can notice that we use a sequential semantics: at each step only a unique rule is executed once. Actually the logic proposed in this paper is very general and can be extended easily to other semantics, e.g., the maximally parallel one.

Definition 2 (Membrane composition)
Let $\Pi = (\mu, w_{j_1}, w_{j_2}, \cdots, w_{j_m}, R_{j_1}, R_{j_2}, \cdots, R_{j_m})$ *be a membrane system and* $i \in Lab - Lab_\Pi$.
We denote by $[\Pi]_i$ *the membrane system*

$$\Pi' = (\mu', w_{k_1}, w_{k_2}, \cdots, w_{k_{m+1}}, R_{k_1}, R_{k_2}, \cdots, R_{k_{m+1}})$$

such that

- μ' *is* μ *enclosed into an external membrane labeled by* i*; the labeling of the membranes of* μ *is preserved in* μ'*;*
- $k_1 = i$ *and* $k_s = j_{s-1}$ *for* $s = 2..m+1$*; consequently* $w_{k_s} = w_{j_{s-1}}$ *and* $R_{k_s} = R_{j_{s-1}}$*;*
- $w_{k_1} = \lambda$, $R_{k_1} = \emptyset$.

Example 1. Consider the membrane system Π defined by

$$\begin{aligned}
\mu &= [\ [\ ]_2\ ]_1;\\
R_1 &= \{a \to b\};\\
R_2 &= \{b \to c\};\\
w_1 &= b;\\
w_2 &= a.
\end{aligned}$$

Then $[\Pi]_3$ is the system Π' defined by

$$\begin{aligned}
\mu' &= [\ [\ [\ ]_2\ ]_1\ ]_3;\\
R'_1 &= R_1 = \{a \to b\};\\
R'_2 &= R_2 = \{b \to c\};\\
R'_3 &= \emptyset;\\
w'_1 &= w_1 = b;\\
w'_2 &= w_2 = a;\\
w'_3 &= \lambda.
\end{aligned}$$

Definition 3 (Parallel composition)
Let $\Pi = (\mu, u_{j_1}, u_{j_2}, \cdots, u_{j_m}, R_{j_1}, R_{j_2}, \cdots, R_{j_m})$ *and* $\Pi' = (\mu', v_{k_1}, v_{k_2}, \cdots, v_{k_n}, R_{k_1}, R_{k_2}, \cdots, R_{k_n})$ *be two membrane systems such that* $j_1 = k_1$ *and* $Lab_\Pi \cap Lab_{\Pi'} = \{j_1\}$*. We call parallel composition of the two systems, denoted by* $\Pi|\Pi'$*, the membrane system*

$$\Pi'' = (\mu'', w_{l_1}, w_{l_2}, \cdots, w_{l_{m+n-1}}, R_{l_1}, R_{l_2}, \cdots, R_{l_{m+n-1}})$$

defined by:

- μ'' *is obtained by enclosing into a common external membrane the contents of the skin membranes of* μ *and* μ'*;*

- *in μ'' the labeling of the membranes in μ and in μ' is preserved; consequently the skin membrane of Π'' is labeled by $l_1 = j_1 = k_1$;*
- $w_{l_1} = u_{j_1} v_{k_1}, R_{l_1} = R_{j_1} \cup R_{k_1}$. [2]

The intuition behind the parallel composition operator is that it can be used to divide an entire membrane system in subsystems, where each subsystem can be recognized/understood by a certain external observer.

Example 2. Consider the membrane systems

$$
\begin{array}{ll}
\Pi : & \Pi' : \\
\mu = [\,[\,]_2\,[\,]_3\,]_1 & \mu' = [\,[\,[\,]_5\,]_4\,]_1 \\
w_1 = ab & w_1 = ee \\
w_2 = cd & w_4 = ccd \\
w_3 = aa & w_5 = a \\
R_1 = \{a \to b,\ a \to c\} & R_1 = \{a \to b, a \to d\} \\
R_2 = \{cd \to a\} & R_4 = \{d \to c\} \\
R_3 = \{a \to b, a \to d\} & R_5 = \{a \to b\}
\end{array}
$$

Then $\Pi|\Pi'$ is the system Π'' defined as

$$
\begin{aligned}
&\mu'' = [\,[\,]_2\,[\,]_3\,[\,[\,]_5\,]_4\,]_1 \\
&w_1 = eeab \\
&w_2 = cd \\
&w_3 = aa \\
&w_4 = ccd \\
&w_5 = a \\
&R_1 = \{a \to b, a \to c, a \to d\} \\
&R_2 = \{cd \to a\} \\
&R_3 = \{a \to b, a \to d\} \\
&R_4 = \{d \to c\} \\
&R_5 = \{a \to b\}.
\end{aligned}
$$

Let denote by $\mathbb{P}_i$ the class of membrane systems having the skin membrane labeled by i. Then it is easy to see that the following theorem holds.

Theorem 1. *$(\mathbb{P}_i, |, [0]_i)$ is an Abelian monoid.*

Also the following theorem can be easily proved.

Theorem 2. *Any membrane system can be composed, by iterating parallel and membrane composition, starting from atomic membrane systems.*

[2] The definition is correct as $Lab_\Pi \cap Lab_{\Pi'} = \{j_1\}$. Notice that, since the labeling of the membranes is preserved, we have that for $s \neq 1$ R_{l_s} and $w_{l_s} = u_{k_s}$ $(w_{l_s} = v_{k_s})$ are preserved as in the original system Π (Π', respectively).

Example 3. Consider the membrane system Π presented in Example 2.

The system Π can be obtained as

$$[\ [cd]_2|[R_2]_2\ ]_1\ |\ [\ [aa]_3|[R_3]_3\ ]_1\ |\ [ab]_1\ |\ [R_1]_1$$

with R_1, R_2 and R_3 as in Π. Clearly $[cd]_2$, $[aa]_3$, $[ab]_1$, $[R_1]_1, [R_2]_2, [R_3]_3$ are atomic membrane systems.

4 Partial Information in Membrane Systems

We want to propose a formal way of playing with *partial information* about a (membrane) system in order to decide some global properties. The idea is to formally describe *open systems*. An open system for an observer is a system formed by a known subsystem and an unknown (opened) part about which the observer does not know anything. So if the observer knows a subsystem S_1 of a bigger system $S_1|S_2$, then the observer considers as entire system, any structure of type $S_1|S_3$, for any possible system S_3. Hence, the properties that the observer knows about the entire system are the properties that systems "like" $S_1|S_2$, $S_1|S_3$, etc. have in common.

Consider again the example, presented in the Introduction, where a virus attacks our body. We have decided to model a relevant part of immune system, say I, in relation with the virus v. Hence the model of a body that has been penetrated by a virus is $body = I|v|S$, where S denotes the rest of the body (we have not considered to model the rest of the body in details since the system I is enough for comprehending the interaction with the virus). Suppose now that the properties we try to specify do not concern only the subsystem $I|v$ (the one we have considered) but the whole body $I|v|S$.

Can we sustain that each property of the system $I|v$ can be stated about the whole body $I|v|S$?

For correctly answering to this question, we propose a logic to play with partial information. Consider a complex biological system about which we have only partial information. This information is collected by some observers placed in different points of the system. Each observer analyzes a subsystem. Our logic develops the framework needed to combine the knowledge of these observers such that is possible to derive interesting properties about the whole system, even without having complete information about it. Playing with observers might cost less then fully investigating the system and it might provide enough information for deciding on the properties we are interested in. All depends on how we place the observers and how we combine their knowledge in deriving complex properties.

Formally, we propose a logic developed in dynamic-epistemic paradigm [9] and enriched with operators from spatial logics [2,3,5,6]. We call it *dynamic epistemic spatial logic*. The syntax allows to express open systems and the knowledge of observers. By combining the knowledge of different observers we can specify and verify complex properties about the whole system without having complete knowledge about it.

In related papers [13,15,12] it has been proposed Hilbert-style axiomatic systems for different such logics, and it has been proved that they are decidable against a semantics based on process algebra, even in the cases for which the classical spatial logics have been proved to be undecidable [4].

5 Playing with Partial Information

In this section we will show how, playing with partial information about a system, we can derive properties of the whole system. For this we reconsider a classical example used in epistemic reasoning [9] adapted for a biologically inspired situation.

Consider a biological system S composed by four disjoint subsystems $S = S_1|S_2|S_3|S_4$. In Figure 1 there are four cells S_1, S_2, S_3 and S_4. Each cell contains a vacuole that can be either normal, having an oval shape as in S_3, S_4, or abnormal having a non-circular shape as the vacuoles of S_1 and S_2. Suppose, in addition, that the system is analyzed by four observers, each observer having access to only a subpart of S. Thus observer O_1 can see the subsystem $S_2|S_3|S_4$, O_2 the subsystem $S_3|S_4|S_1$, O_3 can see the subsystem $S_4|S_1|S_2$ and observer O_4 sees $S_1|S_2|S_3$. Each observer has a display used for making public announcements.

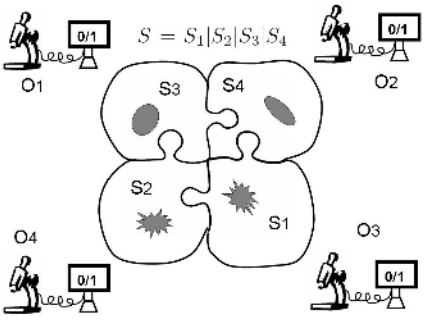


Fig. 1. The system S

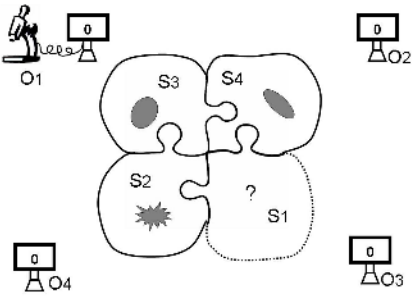


Fig. 2. The perspective of O_1

The observers know that the system S contains abnormal vacuoles and each observer tries to compute the exact number of them and their positions in the system. In doing this the observers do not communicate but only witness the public announcements. Each observer displays 0 until it knows the exact number and positions of abnormal vacuoles, moment in which it switches to 1. In addition, the observers are synchronized by a clock that counts each step of computation. Hence, after each "tick" the observer has to evaluate its knowledge and to decide if its display remains on 0 or switches to 1. Thus each observer computes information about the whole system by using the partial information it possesses and by evaluating the knowledge of the other observers (by reading their displays). If an observer is able to decide the correct number of abnormal vacuoles and their exact positions in the system, then it succeeded to do this with

a lower cost then the cost needed for fully investigating the system. Hereafter we show that such a deduction is possible.

Consider that the real state of the system is the one in Figure 1. And suppose that we can control only the observer O_1. As O_1 sees the subsystem $S_2|S_3|S_4$, it sees an abnormal vacuole in subsystem S_2 and normal vacuoles in S_3 and S_4; in Figure 2 it is represented the perspective of O_1. But it does not know if the system S_1 has a normal or an abnormal vacuole. For O_1 both situations are equally possible. Hence, after the first round of computation the display of O_1 remains to 0. As it concerns observer O_2, it sees an abnormal vacuole, in S_1, but it doesn't know what is in S_2, thus, after the first round, it will show 0 too.

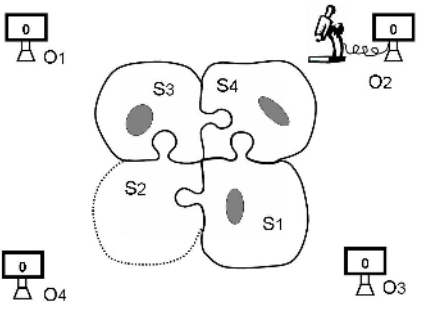


Fig. 3. A hypothetical perspective of O_2

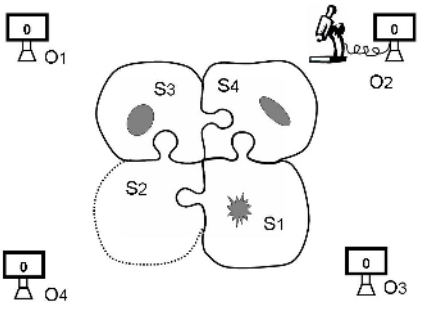


Fig. 4. The real perspective of O_2

It starts the second round of computation. We come back to our observer, O_1. The observer has seen that, after the first round, the observer O_2 has not succeed to understand the situation (as O_2 shows 0 on its display). If the system S_1 would contain a normal vacuole then in the first round O_2 would have seen only normal vacuoles, as in Figure 3. O_2 also knows that there is at least one abnormal vacuole. Hence, if this was the case, O_2 had enough information to decide, in the first round, that the only abnormal vacuole of the system is in S_2. Consequently 1 had to appear on its display. But this was not the case (O_1 can see that by looking to the display of O_2). This means that what O_2 observed was the situation presented in Figure 4. Therefore it is possible to decide that the real situation of the system is the one with an abnormal vacuole in S_1. Thus using *only* O_1 *it is possible to compute the real configuration of the system* and then O_1 will display 1. The example works similarly in more complex situations.

Observe the advantages of this analysis: using only the partial information available to O_1 about the system S and judging the behavior of the other observers, we were able to compute the real configuration of the system. The observers do not exchange information about S, but only about their level of understanding (their observations of) S. The rest can be computed. If each subsystem is very complex, and usually this is the case in biology, then the complete information about the system can be larger then an observer can store or manipulate.

Note also that the observers do not need a central unit for organizing their information. Each observer organizes its own information and makes public announcements about its level of knowledge. They work simultaneously in a distributed network and only playing with their partial information about S and with the information about the state of the network are able to derive overall properties of the system.

The approach fits well with the real situation of biological systems. We work always with partial information which are collected by some observers as results of "measuring" different aspects of a biological phenomena. Sometimes these different "faces" of the same phenomena cannot be integrated in the same mathematical model, or seeking for a property might involve evaluation of different models. For such situations, a formalized way for automatically reasoning, as in the previous example, might help. Hereafter we introduce a logic designed for this purpose.

6 A Logic of Partial Information

As pointed in the previous section, the role of observers in understanding and manipulating biological information is significant. We present a logic of observers, called dynamic epistemic spatial logic [13,14,15,12], developed for specifying and model-checking properties of multi-agent systems. It can be successfully applied for analyzing membrane systems. Our logic proposes a formal way of combining and analyzing the information provided by different observers about the same biological phenomena.

Our logic can be related with spatial logics [3,2,5,6]. For a detailed presentation of it and for a Hilbert-style axiomatization the reader is referred to [13,15,12].

6.1 The Syntax of $\mathcal{L}_{Obs}$

Suppose that we have a class Obs of observers ranged over by A, B, C. We enrich the language of propositional logic with knowledge operators indexed by observers K_A. A statement like $K_A\phi$ is read *"observer A knows ϕ"*. Then we can compose more complex epistemic statements. Thus *"observer A_1 knows that observer A_2 knows ϕ"* is formalized by $K_{A_1}K_{A_2}\phi$. A formula like $K_A\phi \wedge K_A(\phi \to \psi) \to K_A\psi$ is interpreted as *"if observer A knows ϕ and $\phi \to \psi$ then the observer knows ψ"*.

In addition to these operators we add some spatial operators meant to describe the spatial distribution of the subsystems. Anticipating the semantics, we present the intuition behind these operators.

Formula 0 is meant to characterize the trivial membrane system 0 that might be ignored in a complex situation[3].

[3] Some syntaxes of classical logic use 0 for denoting *false*. This is not the case here. We use $\bot$ to denote *false*.

Inspired by spatial logics [2,5,6], we introduce the parallel operator $\phi \parallel \psi$ meant to express the situation in which our system can be decomposed in two (parallel) subsystems, one satisfying ϕ and the other one satisfying ψ.

$\top$ will be satisfied by any system, hence it expresses consistency, "*true*". The role of this element of syntax is essential in expressing open systems. As $\top$ is a property that characterizes any system, $\phi \parallel \top$ characterizes any system that has a subsystem satisfying ϕ and the rest of the system is, possibly, unknown.

By negation, $\bot$ will be used to express the inconsistency.

We also design operators to express membranes. Thus $[\![\phi]\!]_i$ is a property that characterizes a membrane system $[\Pi]_i$ where Π is a membrane system that has the property ϕ. Similarly we introduce formulas $[\![w_i]\!]_i$ and $[\![R_i]\!]_i$ that characterize the atomic membrane systems $[w_i]_i$ and $[R_i]_i$ respectively.

As we propose a logic for the studying of membrane systems together with their evolutions, we allow some modal operators indexed by the transitions of the systems to express the evolutions of a membrane system. Thus $\langle r_i \rangle \phi$ is an operator meant to specify the system Π able to perform a transition r_i, i.e. $\Pi \xrightarrow{r_i} \Pi'$, and Π' satisfies ϕ. These operators are inspired by dynamic logics [10] and are basic operators in Hennessy-Milner logic [11].

Formally, the language of dynamic epistemic spatial logic $\mathcal{L}_{Obs}$ is defined as follows:

Definition 4 (The language). *Let Obs be the set of observers, O an alphabet and Lab a set of labels. We define the language of dynamic epistemic spatial logic, by the following grammar:*

$$\phi := 0 \mid \top \mid \neg\phi \mid \phi \wedge \phi \mid [\![w_i]\!]_i \mid [\![R_i]\!]_i \mid [\![\phi]\!]_i \mid \phi \parallel \phi \mid \langle r_i \rangle \phi \mid K_A \phi$$

where $A \in Obs$, $w \in O^*$, $i \in Lab$ and r_i is a rule of the set R_i.

Definition 5 (Derived operators). *In addition we introduce some derived operators, widely used in dynamic-epistemic logics:*

1. $\bot \stackrel{def}{=} \neg\top$
2. $\phi \vee \psi \stackrel{def}{=} \neg((\neg\phi) \wedge (\neg\psi))$
3. $\phi \rightarrow \psi \stackrel{def}{=} (\neg\phi) \vee \psi$
4. $[r_i]\phi \stackrel{def}{=} \neg(\langle r_i \rangle(\neg\phi))$
5. $1 \stackrel{def}{=} \neg((\neg 0) \parallel (\neg 0))$
6. $\widetilde{K}_A \phi \stackrel{def}{=} \neg K_A \neg\phi$

The dynamic modality $[r_i]\phi$, the dual operator of $\langle r_i \rangle \phi$, captures the weakest precondition of a transition r_i of a membrane system w.r.t. a given post-specification ϕ. We have used the square brackets to denote it, as this notation is classical in dynamic logics (inspired by the box operator of modal logic). It shouldn't be confused with the same brackets use on membrane systems for denoting membrane composition.

Formula 1 is meant to describe the situation in which the system cannot be decomposed into two non-trivial subsystems.

6.2 The Semantics of $\mathcal{L}_{Obs}$

In this subsection we introduce a semantics for the presented logic. It will be defined underpinning on a *satisfiability relation* $\Pi \models \phi$, that establishes the

condition under which we can affirm that the membrane system Π has (satisfies) the property ϕ.

As introduced earlier, each observer sees a membrane system in $\mathbb{P}$. This membrane system is the "structure" that the observer can recognize in any more complex system. Hence, for introducing the semantics, we have to devise an *interpretation function* $int : Obs \longrightarrow \mathbb{P}$ that associates to each observer $A \in Obs$ a membrane system $int(A) = \Pi$ that represents the system that the observer is able to "recognize". The intuition is to define the knowledge of the observer A as the common properties of all systems where A is *active*, i.e., systems that contains Π as subsystem.

Definition 6 (Models and satisfaction). *Given a class Obs of observers and an interpretation function* $int : Obs \longrightarrow \mathbb{P}$ *we introduce the satisfaction relation by:*

$\Pi \models \top$ *always*
$\Pi \models \neg\phi$ *iff* $\Pi \not\models \phi$
$\Pi \models \phi \wedge \psi$ *iff* $\Pi \models \phi$ *and* $\Pi \models \psi$
$\Pi \models \phi \parallel \psi$ *iff* $\Pi = \Pi_1|\Pi_2$ *and* $\Pi_1 \models \phi$, $\Pi_2 \models \psi$
$\Pi \models 0$ *iff* $\Pi = 0$
$\Pi \models [\![w_i]\!]_i$ *iff* $\Pi = [w_i]_i$
$\Pi \models [\![R_i]\!]_i$ *iff* $\Pi = [R_i]_i$
$\Pi \models [\![\phi]\!]_i$ *iff* $\Pi = [\Pi']_i$ *and* $\Pi' \models \phi$
$\Pi \models \langle r_i \rangle \phi$ *iff there exists a transition* $\Pi \xrightarrow{r_i} \Pi'$ *and* $\Pi' \models \phi$
$\Pi \models K_A\phi$ *with* $int(A) = \Pi'$ *iff* $\Pi = \Pi'|\Pi''$ *and* $\forall \Pi'|\Pi''' \in \mathbb{P}$ *we have* $\Pi'|\Pi''' \models \phi$

Then the semantics of the derived operators can be obtained.

$\Pi \models [r_i]\phi$ iff for any Π' such that $\Pi \xrightarrow{r_i} \Pi'$ (if any), $\Pi' \models \phi$
$\Pi \models 1$ iff there are no systems Π', Π'' with $\Pi = \Pi'|\Pi''$ and $\Pi' \neq 0 \neq \Pi''$
$\Pi \models \widetilde{K}_A\phi$ for $int(A) = \Pi'$ iff either $\Pi \neq \Pi'|\Pi''$ for any Π'', or it exists $\Pi'|\Pi'''$ such that $\Pi'|\Pi''' \models \phi$

6.3 Expressivity

Open systems: We can exploit the use of $\top$ to express properties of *open membrane systems*. By an open membrane system we mean a system for which we know only a subpart and we accept any upper-system of the known part as possible configuration for the overall situation. For example if our system is $\Pi = \Pi_1|\Pi_2$ and an observer knows only Π_1, then for the observer any system of type $\Pi_1|\Pi_3$, for any $\Pi_3 \in \mathbb{P}$, is a possible system Π. Hence what is outside Π_1 is "open information" for our observer. Reconsidering the example in section 5, for A_1, in the initial state, Π was an open system because Π_1 has (for A_1) either a normal, or an abnormal vacuole.

If we want to express that a system Π is an open system containing a known subsystem Π_1 then we can express this by allowing an observer $A_1 \in Obs$ to see only Π_1, i.e. $int(A_1) = \Pi_1$. Then $\Pi \models K_{A_1}\top$ means that the system Π is

an upper system of Π_1. Indeed, by our semantics, this means that $\Pi = \Pi_1|\Pi_2$ and for any $\Pi_3 \in \mathbb{P}$ we have $\Pi_1|\Pi_3 \models \top$. But the last condition is trivially true, hence the semantics is equivalent to $\Pi = \Pi_1|\Pi_2$, where Π_2 can be any system. Due to this, we can use $K_{A_1}\top$ to say "in this system Π_1 is a subsystem".

We can be more specific and express that any upper system of Π_1 has the property ϕ. We can do this by taking an upper system of Π_1, say $\Pi = \Pi_1|\Pi_2$, and stating that $\Pi \models K_{A_1}\phi$, where $int(A_1) = \Pi_1$. This is equivalent with saying that for any $\Pi_3 \in \mathbb{P}$ we have $\Pi_3|\Pi_1 \models \phi$.

If we can characterize a membrane system up to identity, we can express that a system Π is an open system containing a known subsystem characterized by ϕ also without using the epistemic operator, by $\Pi \models \phi \parallel \top$. Indeed, w.r.t. our semantics this means that $\Pi = \Pi_1|\Pi_2$ and $\Pi_1 \models \phi$, $\Pi_2 \models \top$. As ϕ satisfies the known system and $\top$ can be stated about any system $\Pi_3 \in \mathbb{P}$ we obtain that any system of type $\Pi_1|\Pi_3$, for any $\Pi_3 \in \mathbb{P}$ satisfies $\phi \parallel \top$.

Characteristic formulas: Using our logic we can define formulas that will fully identify a membrane system. Recall Theorem 2 stating that each membrane system can be decomposed, by using parallel and membrane composition, in atomic membrane systems. We show further how, by induction in top of atomical membrane systems, we can define characteristic formulas for any membrane system.

A characteristic formula of a membrane system Π have to be a formula of our logic ϕ_Π such that

- $\Pi \models \phi_\Pi$
- if $\Pi' \models \phi_\Pi$ then $\Pi' = \Pi$

We define such formulas inductively on structure of Π.

1. $\phi_0 \stackrel{def}{=} 0$
2. $\phi_{[w]_i} \stackrel{def}{=} [\![w]\!]_i$

2′. $\phi_{[R]_i} \stackrel{def}{=} [\![R]\!]_i$

3. $\phi_{\Pi|\Pi'} \stackrel{def}{=} \phi_\Pi \parallel \phi_{\Pi'}$
4. $\phi_{[\Pi]_i} \stackrel{def}{=} [\![\phi_\Pi]\!]_i$

Theorem 3. *Giving an arbitrary membrane system Π, the formula ϕ_Π is a characteristic formula for Π.*

The fact that we can define characteristic formulas for membrane systems open the possibility to project any semantical problem in syntax. Thus, if we want to verify that the system Π has a property ψ, i.e. $\Pi \models \psi$, we can project this problem in syntax where it is equivalent with $\vdash \phi_\Pi \rightarrow \psi$, where we denoted by ϕ_Π the characteristic formula of Π as before. Now the problem $\Pi \models \psi$ is equivalent with proving $\phi_\Pi \rightarrow \psi$ with the axioms of our logic.

Similarly, we can express the fact that between Π and Π' there exists a transition $\Pi \xrightarrow{r_i} \Pi'$ by stating $\vdash \phi_\Pi \rightarrow \langle r_i \rangle \phi_{\Pi'}$. Now $\vdash \phi_\Pi \rightarrow \langle r_i \rangle \phi_{\Pi'}$ can be proved from the axioms iff the transition $\Pi \xrightarrow{r_i} \Pi'$ exists. On this direction we can also imagine more complex situations. Consider, for example, that we have the system Π and we want to know if, after doing the transitions labeled by r_i

then s_j, the t_s it will reach a state (a membrane system) that will have a subpart satisfying ψ. This can be syntactically said by $\vdash \phi_\Pi \rightarrow \langle r_i\rangle\langle s_j\rangle\langle t_s\rangle(\psi \parallel \top)$. Indeed $\psi \parallel \top$ describes a system having a subsystem that satisfies ψ. Then the dynamic operators prefixing it, $\langle r_i\rangle\langle s_j\rangle\langle t_s\rangle(\psi \parallel \top)$, means that the system will reach the state satisfying $\psi \parallel \top$ only after it performs the transitions labeled by r_i, s_j, t_s in this order.

Validity: The presented syntax allows to express the validity of a property in a class of membrane systems having the same external membrane i, i.e. the property is satisfied by any of these systems. We can do this by using a "*blind observer*", i.e. an observer $A' \in Obs$ that sees only the trivial system embedded in i, $int(A') = [0]_i$.

Indeed, the epistemic operator $K_{A'}$ has the following semantics.

$\Pi \models K_{A'}\phi$ iff for any $\Pi'' \in \mathbb{P}_i$ we have $\Pi'' \models \phi$.

This is so because, if a system Π has the property $K_{A'}\phi$ then ϕ is satisfied by any system $\Pi' \in \mathbb{P}$ that can be decomposed in $\Pi' = [0]_i|\Pi''$, i.e. Π' must have the skin membrane i, hence $\Pi' \in \mathbb{P}_i$. But Π' has the property $\Pi'|[0]_i = \Pi'$, as $[0]_i$ is the null element of the monoid $(\mathbb{P}_i, |)$. Hence ϕ is satisfied by any system with the skin i, i.e. it is a valid property over $\mathbb{P}_i$. Thus we can encode, in syntax, the validity of a property.

Consequently, $K_{A'}\top$ is a validity, as $[0]_i$ is a subsystem of any system in $\mathbb{P}_i$, $\Pi = \Pi|[0]_i$.

Satisfiability: Also the satisfiability of a property can be encoded in the syntax. We say that a property is satisfiable if it exists at least one membrane system having this property. For this purpose we use the dual of the knowledge operator for the blind observer $\widetilde{K}_{A'}$ (as before we assume that $int(A') = [0]_i$).

$\Pi \models \widetilde{K}_{A'}\phi$ iff it exists a membrane system $\Pi'' \in \mathbb{P}_i$ such that $\Pi'' \models \phi$

Indeed, if a system Π satisfies $\widetilde{K}_{A'}\phi$ then either $\Pi \neq \Pi'|[0]_i$ (this is not the case as always $\Pi = \Pi|[0]_i$) or it exists Π'' such that $\Pi''|[0]_i \models \phi$. But $\Pi''|[0]_i = \Pi''$, hence it exists a system $\Pi'' \in \mathbb{P}_i$ that satisfies ϕ and vice versa. Thus $\widetilde{K}_{A'}\phi$ provides a way to encode, in syntax, the satisfiability of a property.

6.4 (Some) Axioms, Rules and Theorems

In [13,14,15,12] it has been introduced a Hilbert-style axiomatic system for dynamic epistemic spatial logic. We present further some interesting axioms and theorems that can offer an idea about what can be specified and proved using our logic.

Axiom A 1. $\vdash [\![\phi]\!]_i \parallel [\![0]\!]_i \leftrightarrow [\![\phi]\!]_i$

The previous axioms states that an empty membrane system contained in membrane i do not come with extra properties if it is considered as a subsystem of a system having the skin i. Hence, such a subsystem is "transparent".

Axiom A 2. $\vdash \phi \parallel \psi \rightarrow \psi \parallel \phi$

Axiom A 3. $\vdash (\phi \parallel \psi) \parallel \rho \rightarrow \phi \parallel (\psi \parallel \rho)$

These entail that $\parallel$ organizes an Abelian monoid structure.

Rule R 1. *If* $\vdash \phi \rightarrow \psi$ *then* $\vdash \phi \parallel \rho \rightarrow \psi \parallel \rho$.

This rule establish the monotonicity of parallel composition.

Axiom A 4. $\vdash [r_i](\phi \rightarrow \psi) \rightarrow ([r_i]\phi \rightarrow [r_i]\psi)$.

This axiom is the (K) axiom well-known in modal and dynamic logics which, together with the next rule of necessity shows that, indeed, our operator is an authentic modal operator.

Rule R 2. *If* $\vdash \phi$ *then* $\vdash [r_i]\phi$.

Axiom A 5. $\vdash (\langle r_i \rangle \phi) \parallel \psi \rightarrow \langle r_i \rangle (\phi \parallel \psi)$.

If a subsystem Π_1 of a system $\Pi = \Pi_1 | \Pi_2$ can do a transition r_i and further it satisfies ϕ while its counterpart Π_2 satisfies ψ, then the system Π can be described as able to perform a transition r_i thus passing to a system satisfying $\phi \parallel \psi$.

Axiom A 6. $\vdash K_A\phi \wedge K_A(\phi \rightarrow \psi) \rightarrow K_A\psi$

This axiom A6 is the classical (K)-axiom stating that our epistemic operator is a normal one. It states that if an observer A knows ϕ and that $\phi \rightarrow \psi$ then it knows ψ. It is an usual axiom of knowledge [9].

Axiom A 7. $\vdash K_A\phi \rightarrow \phi$

Also this axiom is classic in modal and epistemic logics – the axiom (T) – necessity axiom. It states that the knowledge of any observer must be true, i.e., an observer cannot know something that is not true.

Axiom A 8. $\vdash K_A\phi \rightarrow K_AK_A\phi$.

Also axiom A8 is well known in epistemic logics. It states that our epistemic agents (observers) have *the positive introspection property*, i.e., if an observer A knows something then it (i.e., the observer) knows that it knows that thing.

Axiom A 9. $\vdash K_A\top \rightarrow (\neg K_A\phi \rightarrow K_A\neg K_A\phi)$

Axiom A9 states a variant of *negative introspection*, saying that if an observer A is active (the system that the observer knows is a subsystem of the whole system) and if the observer does not know ϕ, then the observer knows that does not know ϕ. Negative introspection is also present in classic epistemic logics.

Rule R 3. *If* $\vdash \phi$ *then* $\vdash K_A\top \rightarrow K_A\phi$.

Rule R3 states that any active observer knows all the tautologies. Also in this case we deal with a well known epistemic rule, widely spread in epistemic logics. But our rule works under the assumption that the observer is active.

In [13,15,12] we present a complete axiomatic system and we prove many theorems in it. Hereafter we will sketch some soundness proofs for the previous axioms to clarify the intuitions that motivates the choice of them. Similarly all the axioms can be proved to be sound.

Theorem 4 (Soundness of axiom A5). $\models (\langle r_i \rangle \phi) \parallel \psi \rightarrow \langle r_i \rangle (\phi \parallel \psi)$

Proof. If $\Pi \models (\langle r_i \rangle \phi) \parallel \psi$, then $\Pi = \Pi_1 | \Pi_2$, $\Pi_1 \models \langle r_i \rangle \phi$ and $\Pi_2 \models \psi$. So $\exists \Pi_1 \xrightarrow{r_i} \Pi_1'$ and $\Pi_1' \models \phi$. So $\exists \Pi = \Pi_1 | \Pi_2 \xrightarrow{r_i} \Pi' = \Pi_1' | \Pi_2$ and $\Pi' \models \phi \parallel \psi$. Hence $\Pi \models \langle r_i \rangle (\phi \parallel \psi)$.

Theorem 5 (Soundness of axiom A6). $\models K_A \phi \wedge K_A (\phi \rightarrow \psi) \rightarrow K_A \psi$

Proof. Suppose that $\Pi \models K_A \phi$ and that $\Pi \models K_A (\phi \rightarrow \psi)$, where $int(A) = \Pi_1$. Then $\Pi = \Pi_1 | \Pi_2$ and for any Π' we have $\Pi_1 | \Pi' \models \phi$ and $\Pi_1 | \Pi' \models \phi \rightarrow \psi$. Hence for any such $\Pi_1 | \Pi'$ we have $\Pi_1 | \Pi' \models \psi$ and because $\Pi = \Pi_1 | \Pi_2$ we obtain that $\Pi \models K_A \psi$.

Further we present some meaningful theorems that can be derived with our system.

Theorem 6. $\vdash K_A \phi \rightarrow K_A \top$.

This theorem says that an observer knows something only if it is active.

Theorem 7 (Monotonicity of knowledge). *If* $\vdash \phi \rightarrow \psi$ *then* $\vdash K_A \phi \rightarrow K_A \psi$

The knowledge is monotone, meaning that if a property ϕ guarantees a property ψ then any observer that knows ϕ knows also ψ.

Theorem 8 (Consistency of knowledge). $\vdash K_A \phi \rightarrow \neg K_A \neg \phi$.

This theorem states that the knowledge of an observer is always consistent; the observer cannot know ϕ and $\neg \phi$.

Theorem 9 (Ontological dependency). *If* $int(A) = \Pi_1 | \Pi_2$, $int(A_1) = \Pi_1$ *then* $\vdash K_A \top \rightarrow K_{A_1} \top$.

If the system associated to observer A_1 is a subsystem of the system associated to observer A, then the activation of observer A implies the activation of observer A_1.

For more interesting theorems, the reader is referred to [13,15,12], where, for this logic, it is also developed a semantics on process algebras proved to be sound and complete against the same axiomatic system.

7 A (Simple) Case Study

Consider the membrane system defined as:

Π :	Π' :	Π'' :
$\mu = [\,[\,]_2\,[\,]_3\,[\,]_4\,]_1$	$\mu' = [\,[\,]_2\,[\,]_3\,]_1$	$\mu'' = [\,[\,]_4\,]_1$
$w_1 = \lambda$	$w_1 = \lambda$	$w_1 = \lambda$
$w_2 = a$	$w_2 = a$	$w_4 = c$
$w_3 = b$	$w_3 = b$	
$w_4 = c$		
$R_1 = \{r' : b \longrightarrow b_{in4}\}$	$R_1 = \{r' : b \longrightarrow b_{in4}\}$	$R_1 = \{r' : b \longrightarrow b_{in4}\}$
$R_2 = \{r'' : a \longrightarrow b_{out}\}$	$R_2 = \{r'' : a \longrightarrow b_{out}\}$	$R_4 = \{r^{IV} : b \longrightarrow c_{out}\}$
$R_3 = \{r''' : b \longrightarrow a_{out}\}$	$R_3 = \{r''' : b \longrightarrow a_{out}\}$	
$R_4 = \{r^{IV} : b \longrightarrow c_{out}\}$		

Obviously $\Pi = \Pi'|\Pi''$. Suppose now that we have an observer $A \in Obs$ that can see only the membrane system Π', i.e., $int(A) = \Pi'$. Hence, for such observer, the system Π is an open one, as A can see the subsystem Π' and, for the rest, A accepts any other system as a possible one.

Suppose now that, using the knowledge of A, we want to compute the truth value of the following *property*: if Π contains a membrane 4 then, eventually, it is possible to send an object b to membrane 4 (more exactly after two transitions). We can express this by stating (and proving) that the next formula can be derived, as axiom, from the presented axiomatic system.

$$\vdash K_A\top \rightarrow K_A(\llbracket\llbracket\top\rrbracket_4 \parallel \top\rrbracket_1 \parallel \top \rightarrow \langle r''_2\rangle\langle r'_1\rangle(\llbracket\llbracket b\rrbracket_4 \parallel \top\rrbracket_1 \parallel \top))$$

Indeed, the main precondition $K_A\top$ ensures that the observer A can see something in the system Π (i.e., Π' is a subsystem of Π). This implies that A knows

$$\llbracket\llbracket\top\rrbracket_4 \parallel \top\rrbracket_1 \parallel \top \rightarrow \langle r''_2\rangle\langle r'_1\rangle(\llbracket\llbracket b\rrbracket_4 \parallel \top\rrbracket_1 \parallel \top)$$

We can read the knowledge of A as: if $\llbracket\llbracket\top\rrbracket_4 \parallel \top\rrbracket_1 \parallel \top$, meaning if the membrane 1 contains a membrane 4 and maybe something else then

$$\langle r''_2\rangle\langle r'_1\rangle(\llbracket\llbracket b\rrbracket_4 \parallel \top\rrbracket_1 \parallel \top).$$

The fact that we are not interested in what membrane 4 contains it is expressed by the firsts two $\top$, while the fact that membrane 1 might also contains other things it is specified by the second $\top$.

Now, this post condition can be read as: the system can use the rule r'' in region 2 (which sends an object b to region 1), then it can apply the rule r' in region 1 (because now, in region 1 there is one b) and after doing these two transitions, we obtain a membrane system having membrane 4 inside membrane 1 and region 4 contains the object b. The two $\top$ are used for specifying the fact that in region 4, as well as in region 1, might be also other things in which (in this case) we are not interested in.

Following these steps the specified property can also be proved inside the syntax of the presented logic.

The important point is that we have succeeded to play with partial information *without using a complete description of the system Π, but only using the "point of view" about the system of the observer A.* Moreover, the specified property is true not only for the system Π, but also for any other system which looks to A "indistinguishable" from Π, i.e., any system of type $\Pi'|\Pi'''$ where Π''' is an arbitrary membrane system.

Indeed, if Π''' does not contain membrane 4, then

$$K_A([\![[\![\top]\!]_4 \parallel \top]\!]_1 \parallel \top \rightarrow \langle r_2'' \rangle \langle r_1' \rangle ([\![[\![b]\!]_4 \parallel \top]\!]_1 \parallel \top))$$

is still true, as $[\![[\![\top]\!]_4 \parallel \top]\!]_1 \parallel \top$ is false, and in classic propositional logic false implies anything. From the other side if Π''' contains a membrane 4, then does not matter what else it contains, the system $\Pi'|\Pi'''$ it is still able to send a b in membrane 4 in two transitions, as just shown.

8 Conclusion

The logic we have proposed allows us to specify and formally prove properties of open membrane systems or, in general, properties that involve partial knowledge. Such properties cannot be formally described (in an "easy" way) by using the classic theory of membrane systems. The main idea of the presented logic is that it allows the analysis of the partial knowledge by collecting the partial information collected by observers of a membrane system. As showed in the example presented in Section 5, the logic allows to compute information by using logical reasoning on the information collected by the observers (even if they do not communicate each other). Sometime, using the presented logical tools, it is possible to interpret the "behavior" of the single observers for understanding the information we are looking for. Since this is done is a "distributed" fashion, this type of analysis has a computational price much smaller then the one needed for an analysis of the entire system.

References

1. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter: *Molecular Biology of the Cell*. Garland Publishing, Inc., 2002, fourth edition.
2. L. Caires, L. Cardelli: A Spatial Logic for Concurrency (Part I). *Information and Computation*, 186, 2 (2003).
3. L. Caires, L. Cardelli: A Spatial Logic for Concurrency (Part II). In *Proceedings of CONCUR'2002*, LNCS 2421, Springer-Verlag, 2002.
4. L. Caires, E. Lozes: Elimination of Quantifiers and Decidability in Spatial Logics for Concurrency. In *Proceedings of CONCUR'2004*, LNCS 3170, Springer-Verlag, 2004,
5. L. Cardelli, A.D. Gordon: Anytime, Anywhere: Modal Logics for Mobile Ambients. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, 2000.

6. L. Cardelli, A.D. Gordon: Ambient Logic. *Mathematical Structures in Computer Science*, to appear.
7. M. Cavaliere, P. Leupold: Evolution and Observation: A New Way to Look at Membrane Systems: *Proceedings WMC2003*, LNCS 2933, Springer-Verlag, 2004.
8. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin, 1989.
9. R. Fagin, J.Y. Halpern, Y. Moses, M.Y. Vardi: *Reasoning about Knowledge*. MIT Press, 1995.
10. D. Harel, D. Kozen, J. Tiuryn: *Dynamic Logic*. MIT Press, 2000.
11. M. Hennessy, R. Milner: Algebraic Laws for Nondeterminism and Concurrency. *Journal of ACM*, 32, 1 (1985).
12. R. Mardare: *Logical Analysis of Complex Systems: Dynamic Epistemic Spatial Logics*. PhD Thesis, DIT, University of Trento, Italy, 2006 (available from `http://www.dit.unitn.it/~mardare/publications.htm`).
13. R. Mardare, C. Priami: Decidable Extensions of Hennessy-Milner Logic. *26th International Conference on Formal Methods for Networked and Distributed Systems, FORTE'06*, LNCS, Springer-Verlag, 2006.
14. R. Mardare, C. Priami: *Model Checking Dynamic Epistemic Spatial Logics*. Technical Report DIT-06-009, Informatica e Telecomunicationi, University of Trento, 2006.
15. R. Mardare, C. Priami: *Dynamic Epistemic Spatial Logics*. Technical Report, 03/2006, Microsoft Research Center for Computational and Systems Biology, Trento, Italy, 2006.
16. Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
17. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.
18. The P Systems Web Page: `http://psystems.disco.unimib.it`.

Tau Leaping Stochastic Simulation Method in P Systems*

Paolo Cazzaniga[1], Dario Pescini[1], Daniela Besozzi[2], and Giancarlo Mauri[1]

[1] Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
cazzaniga/pescini/mauri@disco.unimib.it
[2] Università degli Studi di Milano
Dipartimento di Informatica e Comunicazione
Via Comelico 39, 20135 Milano, Italy
besozzi@dico.unimi.it

Abstract. Stochastic simulations based on the τ leaping method are applicable to well stirred chemical systems reacting within a single fixed volume. In this paper we propose a novel method, based on the τ leaping procedure, for the simulation of complex systems composed by several communicating regions. The new method is here applied to dynamical probabilistic P systems, which are characterized by several features suitable to the purpose of performing stochastic simulations distributed in many regions. Conclusive remarks and ideas for future research are finally presented.

1 Introduction

Stochastic modeling is recently gaining more attention in the study of biological systems because "noise" and discreteness play an important role in cellular processes involving few molecules. Many experimental evidences can be found in literature today, such as, e.g., [9,3]. Several examples about stochastic modeling in biological systems, like signal transduction pathways, or the functioning of transcription and translation machinery, can be found in [17,26] and references therein.

It is well known that it is possible to exploit stochastic algorithms to accurately describe the behavior of biological systems, though these approaches lack of computational efficiency. Many new algorithms have been proposed to speed up the computation, trading time for accuracy. A common limitation to many of these approaches is the single volume hypothesis: all the chemical reactions occur within a well mixed single volume at constant temperature and pressure. Here we want to introduce a new stochastic approach in the framework of P systems [20] – exploiting their topological structure and other features – as a novel tool for the modeling of multivolume complex systems. In the following

* Work supported by the Italian Ministry of University (MIUR), under project PRIN-04 "Systems Biology: modellazione, linguaggi e analisi (SYBILLA)".

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 298–313, 2006.

we will assume that the reader is familiar with basic notions on P systems; for further information we refer, e.g., to [21,8] and to the P systems web page: `http://psystems.disco.unimib.it`.

The *stochastic simulation algorithm* (SSA), introduced by Gillespie in [12], is currently used as the reference procedure for performing stochastic and discrete simulations of various biological systems (see, e.g., [1,11,18]). It is an exact numerical simulation method that keeps track of every reaction event occurring in the system. On the other hand, many realistic problems cannot be efficiently solved by using it, since the load of computer work is sometimes very high. To speed up the SSA, Gillespie introduced in [13] the τ *leaping* method as an approximate simulation strategy. Using Poisson random numbers, it is indeed possible to leap over many reaction events in a way that well approximates the exact stochastic simulation.

The SSA, as well as the τ leaping method, are only applicable to well stirred chemical reaction systems within a single fixed volume, at constant temperature. In order to overcome this limit, in this paper we introduce a new method which exploits the structure (formed by several volumes) and the communication features of P systems: using a modified τ leaping procedure, we can then simulate both the behavior of every volume, as well as the behavior of the whole system.

P systems were introduced in [20] as a class of distributed parallel computing devices, inspired by the structure and the functioning of living cells. The basic model consists of a structure composed by several membranes, which delimit regions and can contain objects, evolving in accord to given evolution rules. In one step of a computation, all regions are simultaneously processed by using the rules in a nondeterministic and maximally parallel manner, and at each step all the objects which can evolve should evolve. All the evolved objects are then communicated to the regions specified by a target indication associated with each rule. A computation device is obtained, starting from an initial configuration and letting the system evolve.

Many different classes of P systems as computing devices have been proposed [21]. More recently, P systems have been applied in various research areas, ranging from Biology to Linguistics to Computer Science, see, e.g., [8]. In this paper we use membrane systems as modeling tools and, in particular, we consider *dynamical probabilistic P systems* (DPPs), a stochastic class introduced in [23,24] for the analysis and simulation of the behavior of complex systems. DPPs are discrete and stochastic models, where the probability values associated with the rules change during the evolution of the system. The evolution of DPPs is achieved using a strategy that is similar to the SSA. Moreover, DPPs can be used to tackle the problem of maximal parallelism, a basic feature of P systems which can actually be far from reality in many application cases. Details about DPPs and examples of simulated systems can be found in [22,23,24].

This paper is structured as follows. In Section 2 we recall some τ leaping stochastic algorithms, we describe the τ leaping procedure and show some results in order to test its accuracy and efficiency. In Section 3 we introduce the new τ leaping procedure in the framework of DPPs and we present the results obtained

by the simulation of a benchmark test case. We conclude with some remarks on possible extensions of our work.

2 Gillespie's Stochastic Simulation Methods

In this section we explain how the τ leaping selection procedure works, we present the description of an implementation of the algorithm and we show some results in order to prove the accuracy and the efficiency of this method.

The τ leaping method, first introduced by Gillespie in [13], is used to speed up stochastic simulations where, besides keeping track of every reaction event (as in SSA [12]), one also selects a leap interval where more than one reaction can be fired.

Several improvements of the τ leaping have been proposed by Gillespie and Petzold [14] in order to improve the strategy of selecting the size of the τ leap. Tian and Burrage [25] and Chatterjee *et al.* [7] introduced a *binomial* τ leaping to avoid the possibility of producing negative concentrations of the chemical species. Also Cao *et al.* [4] modified the original τ leaping procedure to work out the negativity problem.

All these forms of τ leaping are lacking in two parts: first, they violate the leap condition [13] since, during the leap, the estimated change of the propensity function is bound by a fraction ε (that is, a pre-specified error control parameter $0 < \varepsilon \leq 1$) over the sum of all propensity functions. In this way, any propensity function that has a relatively small value will be allowed to change by a relatively large amount (the definitions of propensity function and leap condition will be given in Section 2.1). Second, the τ leaping selection requires the evaluation of M^2 auxiliary quantities at each step, where M is the number of reactions in the system.

To avoid these problems, Gillespie *et al.* [5] introduced a new τ selection procedure. This procedure (to which we refer in this paper) is more accurate than the previous ones since it satisfies more closely the leap condition, bounding in a uniform manner the relative changes in the propensity functions. Moreover, it is faster than the previous ones because the number of auxiliary quantities to be computed increases linearly, instead of quadratically, with respect to the number of reactant species.

2.1 Tau Leaping

We recall here the fundamental hypothesis and main definitions needed to describe the τ leaping procedure as presented in [5]. Let $\mathcal{X}$ be a well stirred system in thermal equilibrium consisting of N molecular species $S_1, \ldots, S_N$, which can interact through M chemical reaction channels $R_1, \ldots, R_M$. The vector $\mathbf{X}(t) \equiv (X_1(t), \ldots, X_N(t))$, where $X_i(t)$ is the number of molecules of the species S_i at time t, describes the state of the system at time t. Let $I = \{1, \ldots, N\}$ and $J = \{1, \ldots, M\}$ be, respectively, the sets of indexes over the species and the reaction channels sets.

The probability that a reaction R_j, with $j \in J$, will occur in the next infinitesimal time interval $[t, t + dt)$ in the system state $\mathbf{x} = \mathbf{X}(t)$ is given by $a_j(\mathbf{x})dt$, where $a_j(\mathbf{x})$ is called the *propensity function* of R_j and is defined as $a_j(\mathbf{x}) = h_j(\mathbf{x})c_j$, being $h_j(\mathbf{x})$ the number of distinct reactant molecules combinations and c_j the stochastic rate constant associated to R_j. The changes of species populations are ruled by the *state change vector* $\mathbf{v}_j \equiv (v_{1j}, \ldots, v_{Nj})$, $j \in J$. The element v_{ij} of $\mathbf{v}_j$ represents the multiplicity change of the species S_i due to reaction R_j. Given the above system definition, the τ leaping algorithm can be described as follows.

The aim of the τ leaping procedure is to fire more than one reaction for each time increment $[t, t + \tau)$. The finding of the exact probability distribution of the rules applications, within a generic step of length τ, is a hard task to solve. A possible solution is to approximate the exact behavior of the system, bounding the changes in the reactions propensity functions; this has, as a consequence, a limitation of the time increment. Chosen the τ value, it is then possible to guess the occurring reactions using a Poisson distribution.

Given the state $\mathbf{x}$ of the system $\mathcal{X}$, let $K_j(\tau, \mathbf{x}, t)$ be the exact number of times that a reaction R_j will be fired in the time interval $[t, t + \tau)$, so that $\mathbf{K}(\tau, \mathbf{x}, t)$ is the exact probability distribution vector (having $K_j(\tau, \mathbf{x}, t)$ as elements). For arbitrary values of τ, it is difficult to compute the values of $K_j(\tau, \mathbf{x}, t)$. On the contrary, if τ is small enough that the change in the state during $[t, t + \tau)$ is so slight that no propensity function will suffer an appreciable change in its value (this is called the *leap condition*), then we can evaluate a good approximation of $K_j(\tau, \mathbf{x}, t)$ by using the Poisson random variable with mean and variance $a_j(\mathbf{x})\tau$.

So, starting from the state $\mathbf{x}$ and choosing a value τ that satisfies the leap condition, we can update the state of the system at time $t + \tau$ according to:

$$\mathbf{X}(t + \tau) = \mathbf{x} + \sum_{j=1}^{M} \mathbf{v}_j P_j(a_j(\mathbf{x}), \tau) \tag{1}$$

where $P_j(a_j(\mathbf{x}), \tau)$, for each $j \in J$, denotes an independent sample of the Poisson random variable with mean and variance $a_j(\mathbf{x})\tau$.

Each iterative step of the algorithm is composed by four stages:

1. Generate the maximum changes of each species that satisfy the leap condition.
2. Compute the mean and variance of the changes of the propensity functions.
3. Compute the leap value τ.
4. Toss the reactions to apply.

Hereafter, we describe in detail the motivations and the aims of each of the four stages.

1. Satisfying the leap condition. The procedure for the selection of τ is accomplished in order to bound the relative changes in the molecular populations, in such a way that the relative changes in the propensity functions will be all bounded - during the τ interval - by a small value ε ($0 \leq \varepsilon \leq 1$).

Let $\Delta_\tau X_i$ be the change in the population X_i in the time interval $[t, t+\tau)$. Given the state $\mathbf{x}$ and its projections $x_i = X_i(t)$, the leap condition is:

$$|\Delta_\tau X_i| \;\leq\; max\{\varepsilon_i x_i, 1\} \quad \forall\, i \in I, \tag{2}$$

where the values $\varepsilon_i = \varepsilon_i(\varepsilon, x_i)$ are chosen so that the relative changes in the propensity functions will be all bounded, at least, by ε.

To do that, first determine, for each $i \in I$, the highest order of reaction in which species S_i appears as a reactant (denoted by $HOR(i)$). Then compute:

$$\varepsilon_i = \frac{\varepsilon}{g_i} \tag{3}$$

where $g_i = g_i(x_i)$ is defined as follows:

1. if $HOR(i) = 1$ then $g_i = 1$
2. if $HOR(i) = 2$ then
$$g_i = \begin{cases} 2 & \text{if } R_i : S_i S_k \to \ldots \quad \text{with } i \neq k \\ \left(2 + \dfrac{1}{x_i - 1}\right) & \text{if } R_i : S_i S_i \to \ldots \end{cases}$$
3. if $HOR(i) = 3$ then
$$g_i = \begin{cases} 3 & \text{if } R_i : S_i S_k S_l \to \ldots \quad \text{with } i \neq k \neq l \\ \dfrac{3}{2}\left(2 + \dfrac{1}{x_i - 1}\right) & \text{if } R_i : S_i S_i S_k \to \ldots \quad \text{with } i \neq k \\ \left(3 + \dfrac{1}{x_i - 1}\dfrac{2}{x_i - 2}\right) & \text{if } R_i : S_i S_i S_i \to \ldots \end{cases}$$

The g_i values corresponding to reactions having $HOR > 3$ can be easily computed by taking into account the combinatoric of the species involved in the reactions.

2. Compute mean and variance. To compute the largest value of τ that satisfies the leaping condition (2), we need to evaluate two auxiliary quantities: the mean and the variance of the expected change in the propensity functions.

Referring to the basic τ-leaping formula (1), it is possible to consider the quantity defined in (2) to be:

$$\Delta_\tau X_i = \sum_{j \in J_{ncr}} v_{ij} P_j(a_j(\mathbf{x}), \tau) \quad \forall\, i \in I, \tag{4}$$

where J_{ncr} denotes the set of noncritical reactions.

A *critical reaction* is a reaction with positive propensity function such that a small number of firings is currently left before exhausting one of its reactants. All the other reactions are named, instead, *noncritical reactions*. It is clear that the set of reactions J is the direct sum of the critical J_{cr} and noncritical J_{ncr}

reactions sets: $J \equiv J_{cr} \oplus J_{ncr}$. The motivations of the partition and the choice of $j \in J_{ncr}$ in (4), can be found in [5].

As previously said, the Poisson random variables $P_j(a_j(\mathbf{x}), \tau)$ on the right-hand side of Equation 4 are statistically independent and have mean and variance $a_j(\mathbf{x})\tau$. Hence, the mean and variance of their linear combination can be computed as follows:

$$\langle \Delta_\tau X_i \rangle = \sum_{j \in J_{ncr}} v_{ij}[a_j(\mathbf{x})\tau], \quad var\{\Delta_\tau X_i\} = \sum_{j \in J_{ncr}} v_{ij}^2[a_j(\mathbf{x})\tau] \tag{5}$$

for all $i \in I$. Hence, following the same reasoning that was used in the τ selection introduced in [14], it is possible to consider the bound given in Equation 1 substantially satisfied if it is simultaneously satisfied by the absolute mean and the standard deviation of $\Delta_\tau X_i$:

$$|\Delta_\tau X_i| \leq max\{\varepsilon_i x_i, 1\}, \quad \sqrt{var\{\Delta_\tau X_i\}} \leq max\{\varepsilon_i x_i, 1\}, \tag{6}$$

for all $i \in I$.

Now, substituting formulas (5) into conditions (6) we obtain the following bounds on τ:

$$\tau \leq \frac{max\{\varepsilon_i x_i, 1\}}{|\sum_{j \in J_{ncr}} v_{ij} a_j(\mathbf{x})|}, \quad \tau \leq \frac{max\{\varepsilon_i x_i, 1\}^2}{\sum_{j \in J_{ncr}} v_{ij}^2 a_j(\mathbf{x})} \tag{7}$$

for all $i \in I$.

Finally, it is possible to compute, as described in [14], the two quantities:

$$\mu_i(\mathbf{x}) = \sum_{j \in J_{ncr}} v_{ij} a_j(\mathbf{x}), \quad \forall i \in I, \tag{8}$$

$$\sigma_i^2(\mathbf{x}) = \sum_{j \in J_{ncr}} v_{ij}^2 a_j(\mathbf{x}), \quad \forall i \in I, \tag{9}$$

where we still have the restriction on the noncritical reactions J_{ncr}, due to the conditions of the modified non-negative Poisson τ-leaping [5].

3. Compute the τ value. The leap length is obtained substituting Equations (8, 9) in (7):

$$\tau = \min_{i \in I} \left\{ \frac{max\{\varepsilon x_i / g_i, 1\}}{|\mu_i(\mathbf{x})|}, \frac{max\{\varepsilon x_i / g_i, 1\}^2}{\sigma_i^2(\mathbf{x})} \right\}, \tag{10}$$

where g_i is obtained by Equation 3.

It is also possible to estimate the *mean* $\mu_j(\mathbf{x})\tau$, and the *standard deviation* $\sqrt{\sigma_j^2(\mathbf{x})\tau}$ of the expected change in the propensity function $a_j(\mathbf{x})$ in the time increment τ. Formula 10 requires that these quantities would be bounded by $\varepsilon a_j(\mathbf{x})$ for $j \in J$, thus satisfying the leap condition.

4. Tossing the reactions. The last stage consists in the sampling of random numbers according to the Poissonian distribution $P(a_j(\mathbf{x}), \tau)$ with mean and variance $a_j(\mathbf{x})\ \tau$.

2.2 The Algorithm

In this section, we introduce the algorithm used to compute the value of τ as described in Section 2.1. We recall here that we are considering the system $\mathcal{X}$ with N molecular species interacting through M chemical reaction channels, where the vector $\mathbf{x}$ describes the state of the system and the dynamic is ruled by the state change vectors $\mathbf{v}_j$.

The algorithm works, for each iterative step, as follows:

1. Locate the set of all critical reactions.
2. Compute the quantities μ_i and ${\sigma_i}^2$.
3. Select the value of τ' as indicated in Equation (10).
4. If $\tau' < n/a_0$, where $a_0 = \sum_{j \in J} a_j(\mathbf{x})$, then execute an SSA step as described in [12] and go to step 1, otherwise go to the next step. The factor n is usually set to a reasonable value ($n = 10$ in the following simulations).
5. Compute the sum of the propensity functions of all critical reactions, denoted by $a_0^c(\mathbf{x})$.
6. Generate $\tau'' = 1/a_0^c(\mathbf{x}) \cdot 1/rnd$, where rnd is a value randomly chosen from the uniform distribution over the unit interval $(0, 1)$.
7. If $\tau' < \tau''$ then $\tau = \tau'$, and:
   - For all critical reactions R_j set the number of firings $k_j = 0$.
   - For all noncritical reactions R_j generate k_j as a sample of the Poisson random variable $P(a_j(\mathbf{x}), \tau)$ with mean $a_j(\mathbf{x})\tau$.
8. Else if $\tau'' < \tau'$ then $\tau = \tau''$, and:
   - Select one critical reaction R_j to be fired during this step and set $k_j = 1$; for all other critical reactions R_j set $k_j = 0$.
   - For all noncritical reactions R_j generate k_j as a sample of the Poisson random variable $P(a_j(\mathbf{x}), \tau)$ with mean $a_j(\mathbf{x})\tau$.
9. Update the state of the system: $\mathbf{X}(t + \tau) = \mathbf{X}(t) + \sum_{j \in J} k_j \cdot \mathbf{v_j}$, and check the termination condition $t < t_{max}$.

During step 1, the procedure identifies the set of critical reactions, which will be used in steps 5 and 6 to avoid the possibility to obtain negative multiplicities of the species. In step 2 the quantities needed to obtain the largest value of τ' (step 3) that satisfies the leap condition are computed. If this value (step 4) is less than a multiple of $1/a_0$, then an SSA step is executed because, given the actual state of the system, it is more accurate and efficient than a τ-leap step.

Steps 5 and 6 generate a second candidate leap τ'' that estimates the time of the next critical reaction.

If τ' is smaller than τ'', then some noncritical reactions and no critical reactions will be executed during the leap. Otherwise, several noncritical reactions plus one critical reaction will be executed.

Finally, step 9 updates the state of the system and checks if the current system time t exceeds the prescribed simulation time t_{max}. If the condition holds, terminate the execution, otherwise go to step 1.

2.3 Results

In this section we present some results in order to show the accuracy and efficiency of the τ leaping procedure presented above. We have simulated a simple system of consecutive reactions

$$A \overset{k_1}{\rightarrow} B \overset{k_2}{\rightarrow} C \tag{11}$$

using both the τ leaping method and the SSA, in order to compare the performances of the two procedures.

Figure 1 shows the behavior of the system (11) simulated starting from a population of 1000 individuals of species A; the stochastic constants used for the simulation are $k_1 = 0.1s^{-1}$ and $k_2 = 0.025s^{-1}$. For the simulation with τ-leaping method, $\varepsilon = 0.03$ was used.

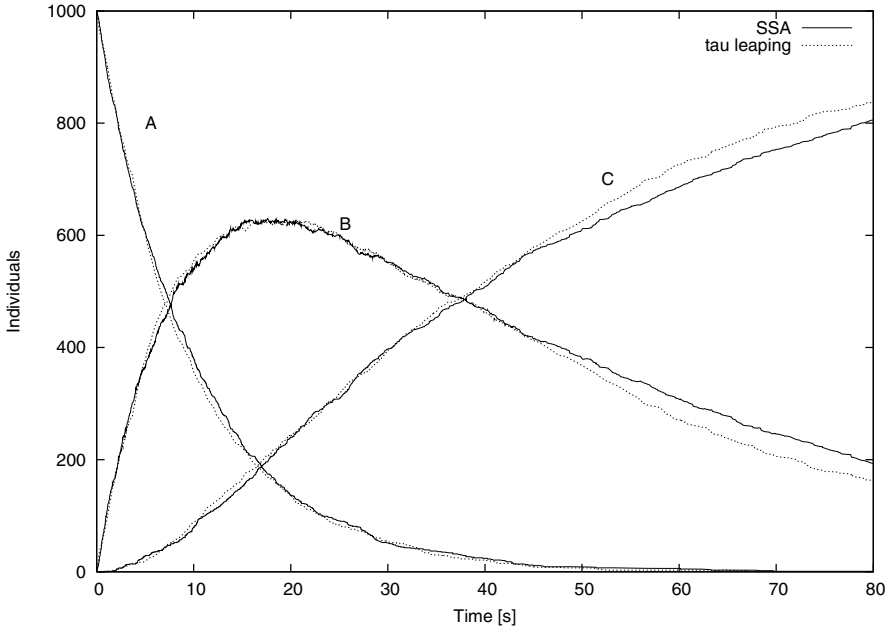


Fig. 1. Consecutive reactions system

Figure 2 shows the histogram plots of the distribution of $A(0.1s)$, that is the number of individuals of species A at time 0.1 s, obtained from 10^6 runs of the SSA and 10^6 runs of the τ leaping method with $\varepsilon = 0.03$.

The similar behaviors of SSA and τ leaping, shown in Figure 1, and the negligible distance between the SSA and the τ leap histograms of Figure 2, prove the accuracy of the τ leaping procedure. The efficiency is proved by the average number of steps of the simulations, which is equal to 102 using SSA and to 79 with the τ leaping method.

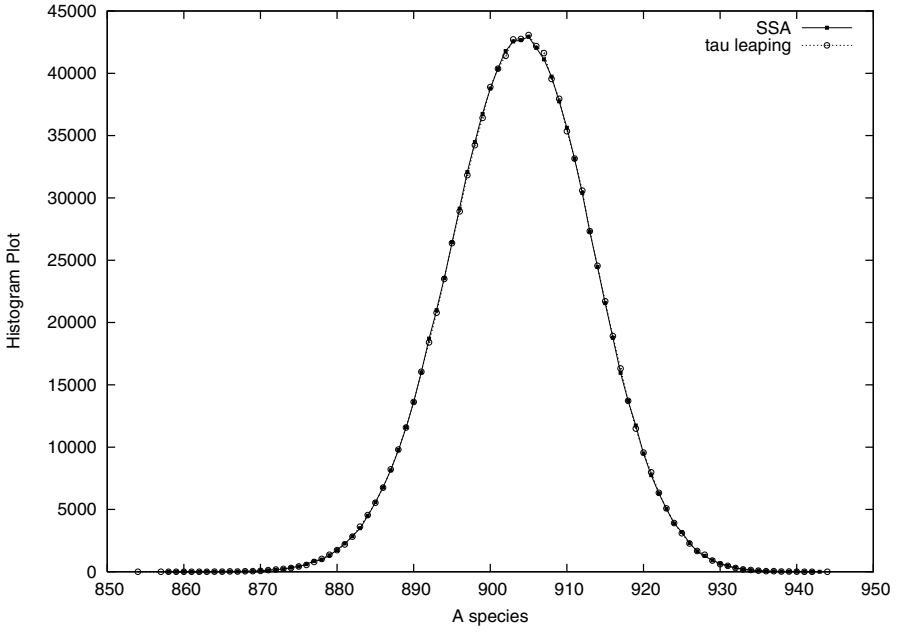


Fig. 2. Histogram plot of the distributions of species A

3 τ-DPPs

Two main problems, in the areas of stochastic modeling and of P systems, motivated our work. The first consists in the fact that SSA, as well as the τ leaping method, are only applicable to well stirred chemical reaction systems contained inside a *single* fixed volume. The second problem concerns DPPs which, up to now, could only allow *qualitative* simulations of a system's dynamics.

A solution to the first problem can be proposed within the framework of P systems, since the membrane structure is suitable to represent systems consisting of *many* regions. Moreover, the communication among regions is a basic and powerful feature in P systems and can be exploited for modeling complex biological systems (such as, e.g., processes involving molecules crossing membranes in cellular systems).

An immediate consequence of the solution to the first problem, that is, the use of P systems for representing multivolume systems and the extension of τ leaping method for such systems, is that DPPs turn out to be valid tools for performing also *quantitative* simulations.

In this section we present the new τ leaping selection method in the framework of DPPs. Then, we describe its implementation and test the exactness of both the procedure and the communication between membranes. In what follows we will refer without distinction to volumes or membranes.

3.1 Tau Leaping Procedure in DPPs

Dynamical probabilistic P systems (DPPs), introduced by Pescini *et al.* in [24], are a stochastic class of P systems where the probability values, associated to each rule according to a prescribed strategy, vary during the evolution of the system. Details about the method for evaluating probabilities, the way the system works and some notes on the corresponding software simulators, as well as for examples of some simulated systems, can be found in [22,23,24,6].

Two major advantages in modeling complex systems by means of DPPs consist in the intrinsic stochasticity and the possibility to probe different levels of parallel rule applications. For instance, it is possible to introduce in DPPs a bounded parallelism by reducing the maximal consumption of objects, at each step, inside all membranes (see, e.g., the use of "mute rules" in [2]). The τ leaping method exploits a "bounded parallelism" as well: the number of reactions applied at each step are dynamically bounded, according to the system state and the underlying process.

The main difference between τ leaping and DPPs, as previously said, is that the first one works on a single volume whereas the second one may simulate complex structured systems, where every membrane can have a different set of rules. Cazzaniga *et al.* reviewed in [6] several stochastic approaches using SSA inside DPPs. The main problem arisen from that study is connected to the communication rules: in order to move objects between membranes, synchronization of all evolving processes has to be forced.

Moreover, DPPs using SSA inside each membrane [6], are parallel at the membranes level but sequential at the rule level: one single rule and its execution time are selected, within each membrane, considering only the internal state of the membrane where the rule will be executed. Therefore, different time increments inside different membranes are obtained. For this reason, different time lines are described although one rule per step (in each membrane where at least one rule can be applied) is executed.

he introduction of τ leaping method inside DPPs works out these problems. First of all, since the same leap of length τ is chosen for all the volumes in the system, the membranes are naturally synchronized. The difference between SSA and τ leaping is that, when using SSA inside DPPs, the synchronization has to be forced at the end of each step, since all volumes generate different τ values (in other terms, after the same number of steps the time simulated within the membranes is different). On the contrary, with τ leaping method we execute the same number of parallel steps, implicitly synchronizing the processes at the end of each step, since the same value of τ is used for all membranes at each iteration.

Secondly, with τ leaping we can consider the communicating rules as the other internal rules, because the rules execution order, within each step, is not important, due to the Poissonian random variable. Moreover, the time needed by the objects to cross the membranes is implicitly taken into account in the rate constants of the communicating rules.

Finally, with τ leaping, we can manage to keep track of the simulated time of the whole system: every membrane of the system evolves according to a common τ value, at each step, and executes the same total number of steps. This is a fundamental feature to simulate complex systems and to quantitatively reproduce their dynamics.

The introduction of τ leaping method inside DPPs requires a new procedure to select a common τ value among all membranes of the system.

We recall here that the original τ leaping procedure can evolve, during each step, in three different manners: (i) like the SSA, executing one reaction during the leap, (ii) executing only noncritical reactions, or (iii) executing noncritical reactions and one critical reaction.

The τ-DPP selection procedure has to consider how every membrane is evolving during the actual step; then, the smallest τ generated within the membranes is used to update the system.

For instance, if a membrane is evolving executing only non critical reactions, but the τ chosen inside it is not the smallest one of the system, then - after receiving the minimal τ from some other membrane - this membrane has to sample the next rules from the set of non critical reactions.

Once the procedure generates a local τ, two different scenarios are possible: no membranes are evolving like SSA, or at least one membrane is evolving according to SSA.

If no membranes are evolving like SSA, the smallest τ (τ_{min}) generated inside the volumes during the current step is chosen. Then the number of firings of the rules is sampled as the Poisson random variable $P(a_j, \tau_{min})$.

If there is at least one membrane evolving in the SSA manner, which generates a value τ_{SSA}, the procedure has to check if $\tau_{min} = \tau_{SSA}$. This requirement is needed because if τ_{SSA} is greater than τ_{min}, it is not possible to apply the rule selected inside that membrane, because the execution would be longer than the leap. Otherwise, $\tau_{min} = \tau_{SSA}$ means that τ_{min} was generated by the membrane evolving according to the SSA, thus the execution of the selected rule is allowed.

3.2 The New Algorithm

In this section we introduce the procedure to select the τ leap value among L membranes, and we show how to execute local and communication rules. The considered structure is composed by l systems $\mathcal{X}_l$, $l = 1, \ldots, L$, each of them defined as in Section 2.1. Moreover, different $\mathcal{X}_l$ can have different sets of rules and object species. The selection of local τ inside the membranes is done following the procedure presented in Section 2.2, the smallest τ of the system is then used to select the number of firings of the rules.

We remark that, in this new version of the algorithm, a $flag$ is used during the iterations to remember how the rule selection has to proceed: $flag = 1$ means that the membrane is evolving according to the SSA, $flag = 2$ means that the membrane has to execute only non critical reactions, and $flag = 3$ means that the membrane will execute non critical reactions and one critical reaction.

For each iterative step, the new version of the algorithm works, inside every volume l, as follows:

1. Locate the set of all critical reactions.
2. Compute the quantities μ_i and ${\sigma_i}^2$.
3. Select the value of τ' as indicated in equation (10).
4. If $\tau' < n/a_0$ then extract an SSA τ as described in [12], set $flag = 1$ and go to step 8, otherwise go to the next step. The factor n is usually set to a reasonable value ($n = 10$ in the following simulations).
5. Compute the sum of the propensity functions of all critical reactions $a_0^c(\mathbf{x})$.
6. Generate $\tau'' = 1/a_0^c(\mathbf{x}) \cdot 1/rnd$, where rnd is a value randomly chosen from the uniform unit interval $(0, 1)$.
7. If $\tau' < \tau''$, then set $\tau = \tau'$ and $flag = 2$, else set $\tau = \tau''$ and $flag = 3$.
8. Receive the smallest τ of the system: τ_{min}.
9. If $flag = 1$ and $\tau = \tau_{min}$, extract one reaction to execute.
10. If $flag = 1$ and $\tau > \tau_{min}$, set $\tau = \tau - \tau_{min}$.
11. If $flag = 2$:
    - For all critical reactions R_j, set the number of firings $k_j = 0$.
    - For all noncritical reactions R_j, generate k_j as a sample of the Poisson random variable $P(a_j(\mathbf{x}), \tau_{min})$ with mean $a_j(\mathbf{x})\tau_{min}$.
12. If $flag = 3$:
    - Select one critical reaction R_j to be fired during this step and set $k_j = 1$, for all other critical reactions R_j set $k_j = 0$.
    - For all noncritical reactions R_j, generate k_j as a sample of the Poisson random variable $P(a_j(\mathbf{x}), \tau_{min})$ with mean $a_j(\mathbf{x})\tau_{min}$.
13. Send and receive objects to and from other membranes (if communication rules were selected).
14. If $flag = 1$ and $\tau > \tau_{min}$ and no objects are received, go to step 8, otherwise go to the next step.
15. Update the state of the system: $\mathbf{X}(t+\tau_{min}) = \mathbf{X}(t) + \sum_{j \in J} k_j \cdot \mathbf{v}_j$, and check the termination condition $t < t_{max}$.

The procedure begins like the pure τ leaping method, that is, the same τ selection is executed (from step 1 to step 7).

After receiving the smallest τ, during step 8, the procedure has to check how the membrane will evolve.

Hence, during step 9, if $flag = 1$ and the internal τ is the smallest of the system, a single rule is applied during the actual iteration (evolving like SSA).

Otherwise (step 10), if $flag = 1$ and the internal τ is greater than the smallest τ of the system, then the value of the local τ is decreased by τ_{min} and no rule is executed. This is necessary because during τ_{min} is not possible to completely execute the rule selected with the SSA, since it would need τ to be executed.

If $flag = 2$ or $flag = 3$, the algorithm selects the rules to fire as a sample of the Poisson random variable $P(a_j(\mathbf{x}), \tau_{min})$ with mean $a_j(\mathbf{x})\tau_{min}$.

All the communication rules are applied during step 13, sending and receiving objects to and from other membranes.

Step 14 is an operation executed, inside a membrane evolving in a SSA manner without applying any reaction, to check if any object has been received. In the positive case a new value of τ will be computed during the next iteration because, although no reactions will be executed, the state of the membrane changes due to the received objects. Otherwise, when no object has been received, in the next iteration the execution of the algorithm inside this membrane jumps directly to step 8 of the algorithm.

Finally, step 15 updates the state of the system and check if the actual system time t exceeds the prescribed simulation time t_{max}. If the condition holds, terminate the execution, otherwise go to step 1.

3.3 A Test Case

To test the new algorithm presented in the previous section, we have implemented the consecutive reactions systems (11) with τ-DPPs. It is possible to test the communication (here considered as instantaneous) between membranes, and check the new τ selection procedure modeling the system by means of two volumes and putting one rule in each volume. We label the membranes with 1 and 2, and then we put rule $A \rightarrow (B, in_2)$ inside volume 1 and $B \rightarrow (C, in_1)$ inside volume 2.

Figure 3 shows that the τ leaping and the τ-DPPs simulations have similar behavior. This benchmark shows that our algorithm is correct and reliable.

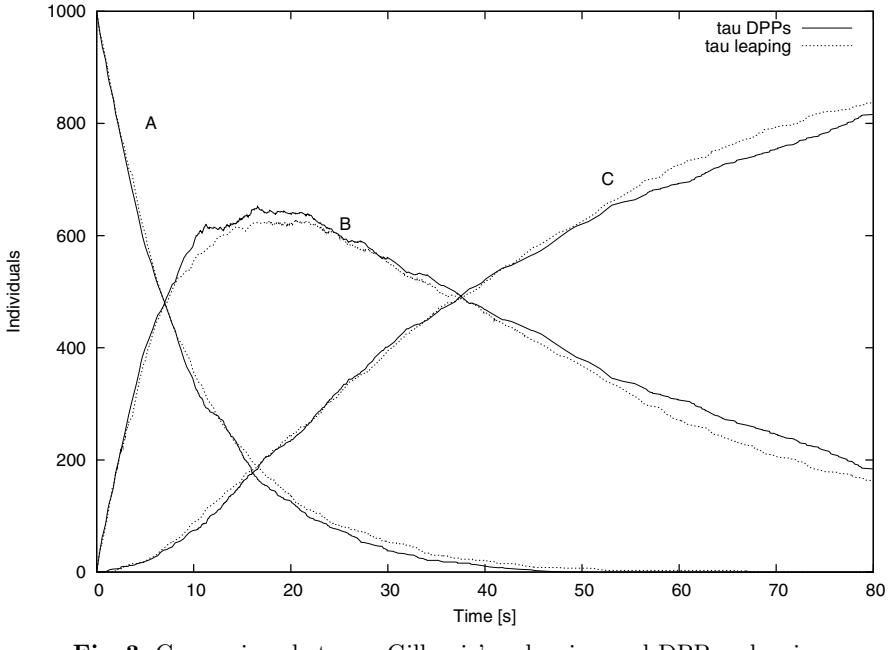


Fig. 3. Comparison between Gillespie's τ leaping and DPPs τ leaping

4 Conclusions

In this paper we have shown how the stochastic method based on the τ leaping procedure can be implemented within the framework of P systems, for the simulation of complex biological systems. In particular, we have considered the class of dynamical probabilistic P systems, to exploit the possibility of modeling systems composed by several volumes and of probing different levels of parallel rule application.

The new τ selection procedure here introduced works by selecting the smallest τ taken from the set of taus generated inside the membranes during the current iteration; then, an evolution step is performed executing several rules, which are selected following the procedure presented in Section 3.

The advantage of introducing τ leaping method inside DPPs is that we can choose the same leap of length τ for all the volumes, we can communicate objects in the right way (assuming that they are sent to the other volumes just at the end of each step, because the execution order does not matter), thus obtaining a good approximation of the system's behavior. An important aspect is that we can trace the simulated time of the whole system, since every membrane evolves according to the chosen common τ value. Moreover, the time needed to run the simulation with the new procedure is shorter than the time needed by the SSA. In Section 3.3 the communication and the new τ leap procedures are tested, comparing the behavior of a simple system implemented both with the single volume model and with the multi-volume model.

Effective and promising results [16,15] have been obtained with the application of τ-DPPs for the stochastic simulations of Ras/cAMP/PKA signalling pathway (in response to glucose addition and intracellular acidification) in *Saccharomyces cerevisiae* [19]. Current applications of τ-DPPs are also addressing the investigation of Repressilator systems [10].

The approach proposed in this paper opens several interesting research lines, ranging from the modeling of real cellular processes or complex biological systems in general, to the algorithmic improvements of the procedure, and the development of other relevant (modelling and simulating) features in the area of Membrane Computing.

References

1. A. Arkin, J. Ross, and H.H. McAdams. Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected Escherichia coli cells. *Genetics*, 149:1633–1648, 1998.
2. D. Besozzi, P. Cazzaniga, D. Pescini, and G. Mauri. Modelling metapopulations with stochastic membrane systems. *Submitted*.
3. W.J. Blake, M. Kærn, C.R. Cantor, and J.J. Collins. Noise in eukaryotic gene expression. *Nature*, 422:633–637, 2003.
4. Y. Cao, D.T. Gillespie, and L.R. Petzold. Avoiding negative populations in explicit Poisson tau-leaping. *Journ. Chem. Phys.*, 123:054104, 2005.

5. Y. Cao, D.T. Gillespie, and L.R. Petzold. Efficient step size selection for the tau-leaping simulation method. *Journ. Chem. Phys.*, 124:044109, 2006.
6. P. Cazzaniga, D. Pescini, F.J. Romero-Campero, D. Besozzi, and G. Mauri. Stochastic approaches in P systems for simulating biological systems. In M.A. Gutiérrez-Naranjo, G. Păun, A. Riscos-Núnez, and F.J. Romero-Campero, Eds., *Proceedings of the Fourth Brainstorming Week on Membrane Computing*, RGNC REPORT 02/2006, 145–164. Fénix Editora, 2006.
7. A. Chatterjee, D.G. Vlachos, and M.A. Katsoulakis. Binomial distribution based tau-leap accelerated stochastic simulation. *Journ. Chem. Phys.*, 122:024112, 2005.
8. G. Ciobanu, G. Păun, and M.J. Pérez-Jiménez, Eds., *Applications of Membrane Computing*. Springer–Verlag, Berlin, 2005.
9. N. Fedoroff and W. Fontana. Small numbers of big molecules. *Science*, 297:1129–1131, 2002.
10. J. Garcia-Ojalvo, M. B. Elowitz, and S. H. Strogatz. Modeling a synthetic multicellular clock: Repressilators coupled by quorum sensing. *PNAS*, 101:10955–10960, 2004.
11. M. Gibbons and J. Bruck. Chemical systems with many species and many channels, *Journ. Phys. Chem.*, 104:1876–1889, 2000.
12. D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journ. Phys. Chem.*, 81:2340–2361, 1977.
13. D.T. Gillespie and L.R. Petzold. Approximate accelerated stochastic simulation of chemically reacting systems. *Journ. Chem. Phys.*, 115:1716–1733, 2001.
14. D.T. Gillespie and L.R. Petzold. Improved leap-size selection for accelerated stochastic simulation. *Journ. Chem. Phys.*, 119:8229–8234, 2003.
15. E. Martegani, R. Tisi, F. Belotti, S. Colombo, C. Paiardi, J. Winderickx, P. Cazzaniga, D. Besozzi, and G. Mauri. Identification of an intracellular signalling complex for RAS/cAMP pathway in yeast: experimntal evidences and modelling. *International Specialised Symposium on Yeasts*, Hanasaari - Espoo, Finland, June 18-21, 2006.
16. E. Martegani, P. Cazzaniga, D. Besozzi, S. Colombo and G. Mauri. Stochastic modeling of the Ras/cAMP signal transduction pathway in yeast. *Computational Methods in Systems Biology*, Trento, Italy, October 18-19, 2006.
17. T.C. Meng, S. Somani, and P. Dhar. Modeling and simulation of biological systems with stochasticity. *In Silico Biology*, 4:0024, 2004.
18. C.J. Morton-Firth. *Stochastic simulation of cell signaling pathways*. PhD thesis, University of Cambridge, Cambridge, UK, 1998.
19. D. Müller, S. Exler, L. Aguilera-Vázquez, E. Guerrero-Martín, and M. Reuss. Cyclic AMP mediates the cell cycle dynamics of energy metabolism in Saccharomyces cerevisiae. *Yeast*, 20:351–367, 2003.
20. G. Păun. Computing with membranes. *J. Comput. Syst. Sci.*, 61:108–143, 2000.
21. G. Păun. *Membrane Computing. An Introduction*. Springer–Verlag, 2002. Berlin.
22. D. Pescini, D. Besozzi, and G. Mauri. Investigating local evolutions in dynamical probabilistic P systems. *Proceedings of Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'05)*. IEEE Computer Press, 440–447, 2005.
23. D. Pescini, D. Besozzi, G. Mauri, and C. Zandron. Analysis and simulation of dynamics in probabilistic P systems. In A. Carbone, N. Pierce, Eds., *DNA Computing, 11th International Workshop on DNA Computing, DNA11*, London, ON, Canada, June 6-9, 2005. LNCS 3892, 236–247, Springer–Verlag, 2006.

24. D. Pescini, D. Besozzi, G. Mauri, and C. Zandron. Dynamical probabilistic P systems. *International Journal of Foundations of Computer Science*, 17:183–204, 2006.
25. T. Tian and K. Burrage. Binomial leap methods for simulating stochastic chemical kinetics. *Journ. Chem. Phys.*, 121:10356–10364, 2004.
26. T.E. Turner, S. Schnell, and K. Burrage. Stochastic approaches for modelling in vivo reactions. *Computational Biology and Chemistry*, 28:165–178, 2004.

P Machines: An Automata Approach to Membrane Computing*

Gabriel Ciobanu[1,2] and Mihai Gontineac[1,3]

[1] Romanian Academy, Institute of Computer Science
Blvd. Carol I nr.8, 700505 Iaşi, Romania
[2] "A.I.Cuza" University, Faculty of Computer Science
[3] "A.I.Cuza" University, Faculty of Mathematics
Blvd. Carol I nr.11, 700506 Iaşi, Romania
gabriel@info.uaic.ro, gonti@uaic.ro

Abstract. In this paper we present P machines corresponding to membrane systems with a single membrane. We give examples of simple P machines for both P systems with promoters and P systems with priorities. For each case we get the same results for both P machines and their corresponding P systems. We present a way of connecting simple P machines, and give an example how the new resulting network corresponds to P systems with more than one membrane.

1 Introduction

Membrane systems (called also P systems) represent a new abstract model of parallel and distributed computing inspired by cell compartments and molecular membranes [10]. A cell is divided in various compartments, each compartment with a specific task, and all of them working simultaneously to accomplish a more general task of the whole system. The membranes of a P system determine regions where objects and evolution rules can be placed. The objects evolve according to the rules associated with each region, and the regions cooperate in order to maintain the proper behavior of the whole system. It is desirable to find good connections with various fields of computer science, including the well-known automata theory. There exist some previous attempts [8,6,9,1] in this direction: [8] and [9] present P automata, namely devices which works mainly with communication rules; [6] presents a P transducer as a form of Mealy membrane automata, but the dynamical aspects are not clearly described; in [1] we find a preliminary study of the dynamics of P systems, and the open problems listed at the end of this paper also motivate our attempt. We have to mention here the existence of some devices working with multisets, namely multiset automata introduced in [7]. All these models are non-compositional.

In [4] we have introduced two versions of Mealy automata, namely Mealy multiset automata, and elementary Mealy membrane automata. We define here an improved version of the elementary Mealy membrane automaton named simple P machine by extending the communication capabilities. We provide some

* This work has been supported by the research grant CNCSIS 1426.

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 314–329, 2006.

examples on how we can use these P machines to describe various features of a membrane system.

2 Mealy Multiset Automata

In order to provide a suitable model for the rules of a membrane, we need the notion of *Mealy multiset automata* (MmA). Roughly speaking, a MmA consists of a *storage location* (a *box* for short) in which we place a multiset over an input alphabet, and a device to translate the input multiset into a multiset over an output alphabet. The way in which a MmA works is described in steps. We have a detection head which detects whether or not a given multiset appears in the input multiset available in the box. If the multiset is detected, then it is removed from the box, and the automaton inserts a multiset over the output alphabet (or a marked symbol if the output alphabet is the same) which cannot be viewed by the detection head. Our automaton stops when no further translation is possible. We say that the submultiset read by the head was translated to a multiset over the output alphabet. We give here only the definitions and the properties which we need for defining the P machines. For more details see [4,5].

Formally, a *Mealy multiset automaton* $\mathcal{A} = (Q, V, O, f, g, q_0)$ is defined by

- a finite set Q of *states*;
- a special state $q_0 \in Q$ which is both initial and final;
- a finite set V of objects representing the *input alphabet*;
- a finite set O of objects representing the *output alphabet*, and $O \cap V = \emptyset$;
- a *state-transition* (*partial*) *mapping* $f : Q \times \mathbb{N}\langle V\rangle \to \mathcal{P}(Q)$;
- an *output* (*partial*) *mapping* $g : Q \times \mathbb{N}\langle V\rangle \to \mathcal{P}(\mathbb{N}\langle O\rangle)$.

If $|f(q, a)| \leq 1$ we say that $\mathcal{A}$ is Q-*deterministic*, and if $|g(q, a)| \leq 1$ we say that $\mathcal{A}$ is O-*deterministic*.

A MmA is endowed with a box where it receives a multiset. After that, it begins to process this multiset over V passing through different *configurations.* It starts with a multiset from $\mathbb{N}\langle V\rangle$, and ends with a multiset from $\mathbb{N}\langle V \cup O\rangle$. A *configuration* of $\mathcal{A}$ is a triple $(q, \alpha, \bar{\beta})$ where $q \in Q$, $\alpha \in \mathbb{N}\langle V\rangle$, $\bar{\beta} \in \mathbb{N}\langle O\rangle$. We say that a configuration $(q, \alpha, \bar{\beta})$ *passes* to $(s, \alpha - a, \bar{\beta} + \bar{b})$ or that we have a *transition* between these configurations, if there is $a \subseteq \alpha$ such that $s \in f(q, a)$ and $\bar{b} \in g(q, a)$. We denote this by $(q, \alpha, \bar{\beta}) \vdash (s, \alpha - a, \bar{\beta} + \bar{b})$. We denote by $\vdash^*$ the reflexive and transitive closure of $\vdash$.

We can alternatively define a configuration to be a pair (q, α) where $\alpha \in \mathbb{N}\langle V \cup O\rangle$, and the transition relation is $(q, \alpha) \vdash (s, \alpha - a + \bar{b})$, with the same conditions as above.

A multiset $\alpha \in \mathbb{N}\langle V\rangle$ is said to be a *totally consumed multiset* (*tc-multiset*) for $\mathcal{A}$ if, starting from a configuration $(q_0, \alpha, \varepsilon)$, the MmA can pass through configurations until it arrives in a configuration $(q_0, \varepsilon, \bar{\beta})$ (i.e., there exists $(q_0, \alpha, \varepsilon) \vdash^* (q_0, \varepsilon, \bar{\beta})$).

A multiset $\alpha \in \mathbb{N}\langle V\rangle$ is said to be a *consumed multiset* (*c-multiset*) for $\mathcal{A}$ if, starting from a configuration $(q_0, \alpha, \varepsilon)$, the MmA can pass through configurations until it arrives in a configuration $(q, \varepsilon, \bar{\beta})$ (i.e., there exists $(q_0, \alpha, \varepsilon) \vdash^* (q, \varepsilon, \bar{\beta})$).

In both these cases, we say also that α was *entirely translated to* $\bar{\beta}$. In all the other situations we say that $\alpha \in \mathbb{N}\langle V\rangle$ is *partially consumed* (*pc-multiset*), or it is *partially translated.*

For the categorical properties of MmA's, as well as MmA's *behavior* and *bisimulation relation*, we refer to [4,5]; in [5] we present some coalgebraic properties of Mealy multiset automata).

2.1 Restricted Direct Product of Mealy Multiset Automata

Let $\mathcal{A}_i = (Q_i, V, O, f_i, g_i)$ be a finite family of Mealy multiset automata, and B_i their corresponding boxes ($i = \overline{1,n}$). We can connect them in *parallel* in order to obtain a new MmA defined by $\mathcal{A} = \bigwedge_{i=1}^{n} \mathcal{A}_i = (\times_{i=1}^{n} Q_i, V, O, f, g)$, called the *restricted direct product* of $\mathcal{A}_i$, where:

- $f((q_1, q_2, \ldots, q_n), a) = (f_1(q_1, a), f_2(q_2, a), \ldots, f_n(q_n, a))$;
- $g((q_1, q_2, \ldots, q_n), a) = (g_1(q_1, a), g_2(q_2, a), \ldots, g_n(q_n, a))$;
- the box B of $\mathcal{A}$ is the disjoint union $\bigsqcup_{i=1}^{n} B_i$ of B_i, $i = \overline{1,n}$;
- a *configuration* of $\mathcal{A}$ is a triple $(q, \alpha, \bar{\beta})$, where $q = (q_1, q_2, \ldots, q_n)$, $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, and $\bar{\beta} = (\bar{\beta}_1, \bar{\beta}_2, \ldots, \bar{\beta}_n)$;
- the (*asynchronous*) *transition relation* of $\mathcal{A}$ is given by $(q, \alpha, \bar{\beta}) \vdash (s, \alpha - a, \bar{\beta} + \bar{b})$ if and only if there is at least an $i \in \overline{1,n}$ such that $s_i \in f_i(q_i, a_i)$ and $\bar{b}_i \in g_i(q_i, a_i)$.

The asynchronous nature of the transition is closer to biology: "The biological systems are massively concurrent, heterogeneous, and asynchronous" [3].

3 Simple P Machines

The previous automata-like approaches for P systems are based on "top-down" communication rules, and the "maximal parallel and nondeterministic" way of applying the rules is more or less visible. We present a new automata-like systems such that every membrane is able to evolve and communicate. While the parallel part is given by the way we connect various MmA's, the "maximality" and the "nondeterministic" aspects are ensured by a "smart" device that is formalized by a *control resource mapping*. The control resource mapping (CRM) is used in both *computing with priorities* and *computing with promoters*. The advantages of such an approach come from the fact that we proceed bottom-up, i.e., we start from a single membrane, then we indicate how we connect single membranes such that these devices can model P systems, as well as tissue-like P systems.

Generally speaking a *simple P machine* (shortly sPM) is built from:

- a *resource box* B together with a *control resource mapping* CRM;
- n Mealy multiset automata, connected in parallel. They consume and translate tc-multisets from their boxes (allocated by CRM) into marked multisets over the same alphabet. We use marked multisets for the output because the input alphabet and the output one must be disjoint; the marking also shows us where the corresponding multiset must go (i.e., all marks belong to a set of targets).

Attached to such a machine we have a *distribution map* denoted by DM. A DM is involved in refreshing the content of the resource box, and in communication with other P machines. We define a cascade product of a P machine with itself in order to go to the next computation step of the P system.

Definition 1. *A* simple P machine *is given by* $\mathcal{M} = (V, \bigwedge_{i=1}^{n} \mathcal{A}_i, O, B, CRM)$, *where:*

- $V = \{a_1, a_2, \ldots, a_m\}$ *is an input alphabet;*
- $\mathcal{A}_i = (Q_i, V, O_i, f_i, g_i)$ *are MmA's connected in parallel;*
- O *is an output alphabet* $O = \bigcup_{i=1}^{n} O_i$ *defined by the output alphabets* $O_i = V \times T_i$, *where target sets* T_i *indicate the indexes of the simple P machines connected to* $\mathcal{M}$; *whenever we consider only an isolated simple P machine, then* $T_i = \{0\}$ *for all* $i = 1, \ldots, n$;
- B *is the box where* $\mathcal{M}$ *receives a multiset for processing;*
- $CRM : \mathbb{N}\langle V\rangle \to \mathcal{P}(\mathbb{N}\langle V\rangle^{n+1})$ *is the* control resource mapping*; this map assigns the resources available in* B *to the bags of the* $\mathcal{A}_i'$*s. It has one supplementary component indicating the multiset unassigned (the remaining multiset). So if* $w = w_1a_1 + w_2a_2 + \cdots + w_ma_m \in \mathbb{N}\langle V\rangle$ *is the input multiset,* $CRM(w)$ *is of the form* $(l_1x_1, l_2x_2, \ldots, l_nx_n, w')$, *where* $\sum_{i=1}^{n} l_ix_i + w' = w$ *and* x_i *are tc-multisets of* $\mathcal{A}_i$.

We have a *distribution mapping* DM attached to a simple P machine. It plays the role of an *interface*. DM is defined on $\mathbb{N}\langle V\rangle \cup \bigcup_{i=1}^{n} \mathbb{N}\langle O_i\rangle$, and takes values in $(\mathbb{N}\langle V\rangle)^T$, where $T = \bigcup_{i=1}^{n} T_i$. DM removes the target component of a multiset, and puts that multiset in the box of the corresponding P machine. A *step of computation* starts with an input multiset, followed by a repartition made by CRM, a translation made by the parallel MmA's (i.e., $\bigwedge_{i=1}^{n} \mathcal{A}_i$), and a distribution done by DM. The distribution conditions satisfied by $CRM(w)$ can define various features of a simple P machine.

Maximal Parallel and Nondeterministic *sPM*

Given a simple P machine $\mathcal{M} = (V, \bigwedge_{i=1}^{n} \mathcal{A}_i, O, B, CRM)$, we define the control resource mapping $CRM : \mathbb{N}\langle V\rangle \to \mathcal{P}(\mathbb{N}\langle V\rangle^{n+1})$ such that this map assigns the resources available in B to the bags of the $\mathcal{A}_i'$s in a maximal parallel and nondeterministic way. If $w = w_1a_1 + w_2a_2 + \cdots + w_ma_m = \sum_{j=1}^{m} w_ja_j \in \mathbb{N}\langle V\rangle$ is the input multiset, then $CRM(w)$ is of the form $(l_1x_1, l_2x_2, \ldots, l_nx_n, w')$ where $\sum_{i=1}^{n} l_ix_i + w' = w$, x_i are tc-multisets of $\mathcal{A}_i$ and $x_i \not\subseteq w'$, for all $i = \overline{1,n}$. In this way we model the maximal parallel feature, because the multiset which remains unprocessed in the box is minimal (does not contain any submultiset that can be processed by a MmA from the direct product). The nondeterminism feature is given by the fact that $CRM(w)$ can take any value from $\{(l_1x_1, l_2x_2, \ldots, l_nx_n, w') \mid \sum_{i=1}^{n} l_ix_i + w' = w,\ x_i$ is a tc-multiset of $\mathcal{A}_i, x_i \not\subseteq w', i = \overline{1,n}\}$. Due to the competition on resources, l_i can take any value between 0 and $max\{l \in \mathbb{N} \mid lx_i \subseteq w\}$. x_i belong to $\mathbb{N}\langle V\rangle$, and so it can be written as a linear combination of a_i's, namely $x_i = \sum_{j=1}^{m} \alpha_{ij}a_j$, $\alpha_{ij} \in \mathbb{N}$. Hence $\sum_{i=1}^{n} l_ix_i =$

$\sum_{i=1}^{n}(\sum_{j=1}^{m} l_i\alpha_{ij}a_j) \subseteq w$, and so $\sum_{j=1}^{m}(\sum_{i=1}^{n} l_i\alpha_{ij})a_j \subseteq \sum_{j=1}^{m} w_j a_j$. This means that for all $j = 1, \ldots, m$ we have $\sum_{i=1}^{n} l_i\alpha_{ij} \leq w_j$.

Lemma 1. *$(l_1, l_2, \ldots, l_n)$ is a solution of the following system:*

$$\sum_{i=1}^{n} l'_i\alpha_{ij} \leq w_j,\ j = \overline{1,m}.$$

In order to obtain a compact writing, we can use the following method of finding the coefficients $l_1, l_2, \ldots, l_n$ for $CRM(w)$ taken from linear algebra:

1. We use a vector $W = (w_1, w_2, \ldots, w_m)$, and the matrix of the tc-multisets of the restricted direct product $\bigwedge_{i=1}^{n} \mathcal{A}_i$, $A = (\alpha_{ij})_{n\times m}$
2. We find the set of *admissible solutions*, i.e., the solutions of the system $L'A \leq W$, where $L' = (l'_1, l'_2, \ldots, l'_n)$. Let $\mathcal{L}^a$ be this set; obviously it is not empty, since $(0, 0, \ldots, 0) \in \mathcal{L}^a$.
3. Choose one $L \in \mathcal{L}^a$, and then calculate $W - LA$;

Theorem 1. *An admissible solution $L = (l_1, l_2, \ldots, l_n)$ is an optimal one (i.e., the simple P machine is working in a <u>maximal</u> parallel way) if and only if every line of matrix* $M_L = \begin{pmatrix} W - LA \\ \ldots \\ W - LA \end{pmatrix}_{n\times m} - A$ *has at least a negative element.*

Proof. If $(l_1, l_2, \ldots, l_n)$ is a solution, it must satisfy the maximality condition, i.e., for all x_k, $x_k \not\subseteq w' = w - \sum_{i=1}^{n} l_i x_i$. It follows that for all $k = \overline{1,n}$, $x_k \not\subseteq \sum_{j=1}^{m} w_j a_j - \sum_{i=1}^{n} l_i x_i$ iff $\sum_{j=1}^{m} \alpha_{kj}a_j \not\subseteq \sum_{j=1}^{m} w_j a_j - \sum_{i=1}^{n}(\sum_{j=1}^{m} l_i\alpha_{ij}a_j) = \sum_{j=1}^{m}(w_j - \sum_{i=1}^{n} l_i\alpha_{ij})a_j$. Thus, if $(l_1, l_2, \ldots, l_n)$ is an optimal solution for all x_k, $k = \overline{1,n}$, then there is a $j_k \in \overline{1,m}$ such that $\alpha_{kj_k} > w_{j_k} - \sum_{i=1}^{n} l_i\alpha_{ij_k}$, i.e., $w_{j_k} - \sum_{i=1}^{n} l_i\alpha_{ij_k} - \alpha_{kj_k} < 0$. This means that every line of matrix M_L has at least a negative element. □

Maximal Consuming and Nondeterministic *sPM*

We refer here to a maximal consuming P system, namely a maximal parallel P system which consumes the largest number of resources among all the possibilities. Considering a simple P machine $\mathcal{M} = (V, \bigwedge_{i=1}^{n} \mathcal{A}_i, O, B, CRM)$, the maximal parallelism and nondeterminism is given by $CRM(w) \in \{(l_1x_1, l_2x_2, \ldots, l_n x_n,\ w') \mid \sum_{i=1}^{n} l_i x_i + w' = w,\ x_i \text{ is a tc-multiset of } \mathcal{A}_i, x_i \not\subseteq w', i = \overline{1,n}\}$. Due to the competition on resources, l_i can take any value between 0 and $max\{l \in \mathbb{N} \mid lx_i \subseteq w\}$. x_i belong to $\mathbb{N}\langle V\rangle$, so it can be written as a linear combination of a_i, namely $x_i = \sum_{j=1}^{m} \alpha_{ij}a_j$, $\alpha_{ij} \in \mathbb{N}$. Hence $l_i x_i = \sum_{j=1}^{m} l_i\alpha_{ij}a_j$. Since $l_i x_i \subseteq w = \sum_{j=1}^{m} w_j a_j$, we obtain that $l_i\alpha_{ij} \leq w_j$ for all $j = 1, \ldots, m$. The difference with respect to a maximal parallel approach is given by an additional *objective function* $\Phi : \mathbb{N}^n \to \mathbb{N}\langle V\rangle$ given by $\Phi(l'_1, l'_2, \ldots, l'_n) = \sum_{i=1}^{n} l'_i x_i$.

Theorem 2. *A simple P machine is working in a maximal consuming and nondeterministic way if and only if for any given input multiset w, the coefficients l_i of $CRM(w)$ are solutions of the following problem of integer programming:*

$$\begin{cases} \sum_{i=1}^{n} l'_i \alpha_{ij} \leq w_j & j = \overline{1,m} \\ l'_i \geq 0 & i = \overline{1,n} \\ max\, \Phi(l'_1, l'_2, \ldots, l'_n) \end{cases}.$$

Example 1. We give here a simple example.

$$\mathcal{M} = (\{a,b,c\}, \bigwedge_{i=1}^{2} \mathcal{A}_i, \{(a,0),(b,0),(c,0)\}, B, CRM),$$

with $TC(\mathcal{A}_1) = \{n(a+b) | n \in \mathbb{N}\}$, $TC(\mathcal{A}_2) = \{n(a+c) \mid n \in \mathbb{N}\}$. $\mathcal{A}_1$ translates $l(a+b)$ into $l(c,0)$, and $\mathcal{A}_2$ translates $l(a+c)$ into $l(b,0)$. Let us consider the input multiset $w = 3a+2b+2c$. Then $CRM(w) \in \{(2(a+b), a+c, c), (a+b, 2(a+c), b)\}$. Simple calculations lead us to the output of the machine which can be either $(b,0) + 3(c,0)$ or $3(b,0) + (c,0)$.

Simple P Machines with Promoters

In P machines with promoters, some of the objects of the input alphabet of arbitrary $\mathcal{A}_i$ are special elements called *promoters*. Only their presence in the resource box allows the corresponding $\mathcal{A}_i$ to receive the related resources in its input bag. If $V = \{a_1, a_2, \ldots, a_m\}$, we consider that the last $m-p$ elements are promoters, and denote their set by P, i.e., $P = \{a_{p+1}, a_{p+2}, \ldots, a_m\}$. We denote by P_i the set of promoters of $\mathcal{A}_i$; it can be empty if $\mathcal{A}_i$ has not promoters.

Let $w = w_1 a_1 + w_2 a_2 + \cdots + w_m a_m \in \mathbb{N}\langle V \rangle$ the input multiset of the sPM $\mathcal{M} = (V, \bigwedge_{i=1}^{n} \mathcal{A}_i, O, B, CRM)$ with promoters in P. If $P_i = \{a_{i_1}, a_{i_2}, \ldots, a_{i_t}\}$, then $\mathcal{A}_i$ can receive from CRM an input multiset in its bag if and only if $w_{i_1}, w_{i_2}, \ldots, w_{i_t}$ are all different from 0, i.e., $\prod_{s=1}^{t} w_{i_s} \neq 0$. As usual, we have that $CRM(w) \in \{(l_1 x_1, l_2 x_2, \ldots, l_n x_n, w') \mid \sum_{i=1}^{n} l_i x_i + w' = w\}$. An approach similar to the previous subsections leads us to the following result:

Proposition 1. *$\mathcal{A}_i$ receives an input multiset in its bag iff $(\star)$ $\prod_{a_{i_s} \in P_i} w_{i_s} \neq 0$. Moreover, the coefficients of the tc-multisets for all the $\mathcal{A}'_i s$ which satisfy the previous condition $(\star)$ are obtained as solutions of the system:*

$$\sum_{i \in I} l'_i \alpha_{ij} \leq w_j, j = \overline{1,m},$$

where we denote by $I \subseteq \{1,2,\ldots,n\}$ the set of indices of specific $\mathcal{A}'_i s$ where the condition $(\star)$ is satisfied.

Simple P Machines with Priorities

Let $\mathcal{M} = (V, \bigwedge_{i=1}^{n} \mathcal{A}_i, O, B, CRM)$ be a simple P machine. We consider a *partial priority order relation* over the set of the Mealy multiset automata that are part of $\mathcal{M}$, and we denote this priority order by " $\leq$ ". The possible values of CRM for an input multiset $w = \sum_{j=1}^{m} w_j a_j$ are more difficult to be obtained, at least from the formal point of view. We define a relation of *immediate precedence*:

$\mathcal{A}_i \preceq \mathcal{A}_j$ if $\mathcal{A}_i \leq \mathcal{A}_j$ and there is no $\mathcal{A}_k$ such that $\mathcal{A}_i < \mathcal{A}_k < \mathcal{A}_j$.

This relation defines some *levels of precedence* given by a partition of the set $\{1, 2, \ldots, n\}$ of MmA's indices. The levels of precedence are defined by:

level 0: $\mathcal{L}_0$ contains the indices of the MmA's which have no predecessor

...

level $(s+1)$: $\mathcal{L}_{s+1} = \{j \in \overline{1,m} \mid (\exists) r_j \in L_s$ such that $\mathcal{A}_{r_j} \prec \mathcal{A}_j\}$

...

Suppose now that $\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_t$ are the levels of precedence. As in the nondeterministic and maximal parallel case, we denote by $A = (\alpha_{ij})_{n \times m}$ the matrix of the tc-multisets of the restricted direct product $\bigwedge_{i=1}^{n} \mathcal{A}_i$. With respect to the levels of precedence, we denote by A_s the matrix obtained from A by erasing all its lines which do not have indexes from $\mathcal{L}_s$. We apply a similar procedure for the unknown coefficients, denoting by L'_s the vector obtained from $L' = (l'_1, l'_2, \ldots, l'_n)$ by removing the components which have not indexes from $\mathcal{L}_s$. We obtain the possible values for $CRM(w)$ by solving the following set of problems:

Level $\mathcal{L}_0$

Step 0.1. Find the set of *admissible solutions* for level $\mathcal{L}_0$, i.e., the solutions of the system $L'_0 A_0 \leq W$. Let $\mathcal{L}_0^a$ be this set (obviously it is non-empty because $(0, 0, \ldots, 0) \in \mathcal{L}_0^a$).

Step 0.2. For all $L_0 \in \mathcal{L}_0^a$, calculate $W - L_0 A_0$, and then consider a matrix

$$M_{L_0} = \begin{pmatrix} W - L_0 A_0 \\ \ldots \\ W - L_0 A_0 \end{pmatrix} - A_0.$$

Step 0.3. If every line of M_{L_0} contains at least a negative element, the corresponding admissible solution L_0 is an optimal one, and so it can be chosen. We have a nondeterministic choice, since it can have several optimal solutions. Let $w^0 = w - \sum_{i \in L_0} l_i x_i$ such that $w^0 = \sum_{j=1}^{m} w_j^0 a_j$. We denote by W^0 its corresponding vector.

Level $\mathcal{L}_1$

Step 1.1. Find the set of *admissible solutions* for level $\mathcal{L}_1$, i.e., the solutions of the system $L'_1 A_1 \leq W^0$. Let $\mathcal{L}_1^a$ be this set (obviously it is non-empty because $(0, 0, \ldots, 0) \in \mathcal{L}_1^a$).

Step 1.2. For all $L_1 \in \mathcal{L}_1^a$, calculate $W^0 - L_1 A_1$, and then consider a matrix

$$M_{L_1} = \begin{pmatrix} W^0 - L_1 A_1 \\ \ldots \\ W^0 - L_1 A_1 \end{pmatrix} - A_1.$$

Step 1.3. If every line of M_{L_1} contains at least a negative element, the corresponding admissible solution L_1 is an optimal one, and so it can be chosen. We have a nondeterministic choice, since it can have several optimal solutions. Let $w^1 = w^0 - \sum_{i \in L_1} l_i x_i$ such that $w^1 = \sum_{j=1}^{m} w_j^1 a_j$. We denote by W^1 its corresponding vector.

...

Suppose that we have $\{l_i, i \in L_r\}$, $w^r = w^{r-1} - \sum_{i \in L_r} l_i x_i$, $w^r = \sum_{j=1}^{m} w_j^r a_j$, and we denote by W^r its corresponding vector.

Level $\mathcal{L}_r$:

Step r.1. Find the set of *admissible solutions* for level $\mathcal{L}_{r+1}$, i.e., the solutions of the system $L'_{r+1}A_{r+1} \leq W^r$. Let $\mathcal{L}^a_{r+1}$ be this set (obviously it is non-empty because $(0, 0, \ldots, 0) \in \mathcal{L}^a_{r+1}$).

Step r.2. For all $L_{r+1} \in \mathcal{L}^a_{r+1}$, calculate $W^r - L_{r+1}A_{r+1}$, and then consider the matrix $M_{L_{r+1}} = \begin{pmatrix} W^r - L_{r+1}A_{r+1} \\ \ldots \\ W^r - L_{r+1}A_{r+1} \end{pmatrix} - A_{r+1}$.

Step r.3. If every line of $M_{L_{r+1}}$ contains at least a negative element, the corresponding admissible solution L_{r+1} is an optimal one, and so it can be chosen.

Solving the existing levels of precedence, we get the possible values of $CRM(w)$.

4 Examples of Simple P Machines

We describe first how priorities and promoters are useful in defining arithmetical operations in membrane systems. After each example we provide the simple P machine describing the corresponding membrane systems.

4.1 Control Mechanisms in P Systems

In P systems several mechanisms allow to control the computation. In general the objects and the rules governing the computation are chosen in a nondeterministic way. Moreover, this choice is exhaustive in the sense that no rule can be further applied in the same evolution step: this is the maximal parallel rewriting. A global clock is assumed, that is the same clock for all the regions of a membrane system. At each tick of this clock, a current configuration of the system is transformed into another one, and so defining a transition between the configurations of the system. A sequence of transitions is called a computation. A computation is halting if it reaches a halting configuration, one where no rules are applicable at all.

We have various control mechanisms in membrane systems. They are inspired by some biological entities. For instance, we have catalysts representing objects which appear on both left-hand and right-hand sides of a rule. The catalysts directly participate in rules (but are not modified by them), and they are counted as any other object such that the number of applications of a rule involving a catalyst is as large as the number of copies of the catalyst. They can be used to apply a certain rule in a sequential way, increasing the control of using the rule. Another controlling mechanism is given by activators, a formal representation of enzymes. An activator is related to a rule. The rules need activators to be applied, so the parallelism of each rule is limited to the number of its activators. The activators can evolve in the same step (this is not possible for catalysts).

Here we refer mainly to control mechanisms defined over sets of rules rather than individual rules; such mechanisms are given by priorities and promoters. A priority relation among rules means that in each region we have a partial order

relation on the set of rules, and a rule can be chosen (to process a multiset of objects) only if no rule of a higher priority is applicable in the same region. Promoters and inhibitors formalize the reaction enhancing and reaction prohibiting roles of various substances (molecules) present in cells. In membrane systems, promoters and inhibitors are represented as multisets of objects associated with given sets of rules. A rule from such a set of a given region can be used only if all the promoting objects are present, and all the inhibiting objects are not present in that region. From the generative point of view, there is a symmetry between the two ideas: systems with promoters are equal in power to systems with inhibitors, and they characterize the recursively enumerable sets of natural numbers. Membrane systems with promoters/inhibitors achieve universal computations in a simpler way. From a technical point of view, it is much easier to work with promoters. The systems become simpler; if we have enough promoters, then systems with only one membrane are already universal.

Regarding the difference between promoters and catalysts, we can say that the catalysts directly participate in rules, and they are counted as objects required by rules, and the number of rule application in parallel is as large as the number of catalysts. In the case of promoters, the presence of only one promoter makes it possible to use a rule involving that promoter as many times as possible, without any restriction.

4.2 Membrane Systems and Their Corresponding *sPM*

We present some examples of P systems implementing arithmetic operations on numbers represented by the numbers of objects. In these examples we use priorities and promoters as control mechanisms in membrane computing, presenting membrane systems with priorities and promoters for multiplication. Other arithmetical operations on numbers represented by using unary and binary compact encodings are presented in [2].

Multiplication with promoters: Figure 1 presents a P system Π_1 with promoters for multiplication of n (objects a) by m (objects b), the result being the number of objects d in membrane 0. The object a is a promoter in the rule $b \to bd|_a$, i.e., this rule can only be applied in the presence of object a. The available m objects b are used in order to apply m times the rule $b \to bd|_a$ in parallel; based on the availability of a objects the rule $au \to u$ where u is a catalyst is applied in the same time and consumes an a. The procedure is repeated until no object a is present within the membrane. Note that each time when one object a is consumed, then m objects d are generated.

$$\begin{aligned}
&\Pi_1 = (V, \mu, w_0, R_0, 0),\\
&V = \{a, b, d, u\},\ \mu = [_0]_0,\ w_0 = a^n b^m u,\\
&R_0 = \{r_1 : b \to bd|_a,\ r_2 : au \to u\}.
\end{aligned}$$

We describe now a simple P machine computing the multiplication presented by the previous P systems. We have only one membrane, and we denote it

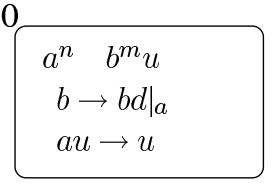


Fig. 1. Multiplication with promoters

with 0; therefore the target can be only 0. $\mathcal{M} = (V, \mathcal{A}_1 \bigwedge \mathcal{A}_2, O, B, CRM)$ with the input alphabet is $V = \{a, b, c, d, u\}$ and the output alphabet is $O = \{(a, 0), (b, 0), (c, 0), (d, 0), (u, 0)\}$.

$\mathcal{A}_1$ translates b into $(b, 0) + (d, 0)$, $\mathcal{A}_2$ translates $a + u$ into $(u, 0)$.

CRM is defined as

$CRM(k_1a + k_2b + k_3u + k_4d) = (l_1b, l_2(a + u), w')$, where

$l_1b + l_2(a + u) + w' = k_1a + k_2b + k_3 + k_4d$, and

$l_1 = \begin{cases} k_2 \text{ if } k_1 \neq 0 \\ 0 \text{ if } k_1 = 0 \end{cases}$, $l_2 = min\{k_1, k_3\}$.

The distribution mapping is defined by

$DM(l_1((d, 0) + (b, 0)) + l_2(u, 0)) = w' + l_1b + l_1d + l_2u$,

where w' is the content of B before applying the resource distribution.

Proposition 2. *$\mathcal{M}$ computes the product of two positive integers.*

Proof. We insert initially $na+mb+u$ in the resource box. We have the following sequence of computations:

$na + mb + 1u \Longrightarrow_{CRM} (mb, a + u, (n - 1)a) \Longrightarrow_{\mathcal{A}_1 \bigwedge \mathcal{A}_2}$
$(m((b, 0) + (d, 0)), (u, 0), (n - 1)a) \Longrightarrow_{DM} (n - 1)a + mb + u + md \Longrightarrow_{CRM}$
$(mb, a + u, (n - 2)a + md) \Longrightarrow_{\mathcal{A}_1 \bigwedge \mathcal{A}_2}$
$(m((b, 0) + (d, 0)), (u, 0), (n - 2)a + md) \Longrightarrow_{DM} (n - 2)a + mb + u + 2md$
$\ldots$
$\Longrightarrow_{CRM} (mb, a + u, a + (n - 2)md) \Longrightarrow_{\mathcal{A}_1 \bigwedge \mathcal{A}_2}$
$(m((b, 0) + (d, 0)), (u, 0), a + (n - 2)md) \Longrightarrow_{DM} a + mb + u + (n - 1)md$
$\Longrightarrow_{CRM} (mb, a + u, (n - 1)md) \Longrightarrow_{\mathcal{A}_1 \bigwedge \mathcal{A}_2}$
$(m((b, 0) + (d, 0)), (u, 0), (n - 1)d) \Longrightarrow_{DM} mb + u + nmd.$ □

Multiplication with priorities: Figure 2 presents a membrane system Π_2 with priorities for multiplication of n (objects a) by m (objects b), the result being the number of objects d in membrane 0.

$$\Pi_2 = (V, \mu, w_0, (R_0, \rho_0), 0),$$
$$V = \{a, b, d, e, u, v\}, \mu = [_0]_0, w_0 = a^n b^m u,$$
$$R_0 = \{r_1 : bv \to dev,\ r_2 : av \to u,\ r_3 : eu \to dbu,\ r_4 : au \to v\},$$
$$\rho_0 = \{r_1 > r_2,\ r_3 > r_4\}.$$

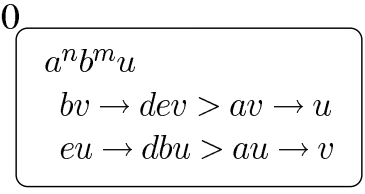


Fig. 2. Multiplication with priorities

We use the priority relation between rules; for instance $bv \to dev$ has a higher priority than $av \to u$, meaning the second rule is applied only when the first one cannot be applied anymore. Initially only the rule $au \to v$ can be applied, generating a catalyst v which activates m times the rule $bv \to dev$. Then $av \to u$ consumes an a, and transform the catalyst v into a catalyst u. Now $eu \to dbu$ is applied m times, followed by another change of catalyst u into a catalyst v by consuming an a (this is done by the rule $au \to v$). The procedure is repeated until no object a is present within the membrane. It is easy to note that each time when one object a is consumed, then m objects d are generated.

We describe now a simple P machine that does the same job. We have only one membrane denoted by 0, and the target can be only 0. We have $\mathcal{M} = (V, \bigwedge_{i=1}^{4} \mathcal{A}_i, O, B, CRM)$, where the alphabets are $V = \{a, b, c, d, e, u, v\}$ and $O = \{(a,0), (b,0), (c,0), (d,0), (e,0), (u,0), (v,0)\}$.

$\mathcal{A}_1$ translates $e + u$ into $(d,0) + (b,0) + (u,0)$, $\mathcal{A}_2$ translates $a + u$ into $(v,0)$, $\mathcal{A}_3$ translates $b + v$ into $(d,0) + (e,0) + (v,0)$, and $\mathcal{A}_4$ translates $a + v$ into $(u,0)$. CRM is defined as $CRM(k_1a + k_2b + k_3d + k_4e + k_5u + k_6v)=$

$=(l_1(e+u), l_2(a+u), l_3(b+v), l_4(a+v), w')$, where

$l_1(e+u)+l_2(a+u)+l_3(b+v)+l_4(a+v)+w' = k_1a+k_2b+k_3c+k_4d+k_5e+k_6f$, and

$l_1 = min\{k_4, k_5\}$,

$l_2 = \begin{cases} 0 & l_1 \neq 0 \\ min\{k_1, k_5\} & l_1 = 0 \end{cases}$,

$l_3 = min\{k_2, k_6\}$,

$l_4 = \begin{cases} 0 & l_3 \neq 0 \\ min\{k_1, k_6\} & l_3 = 0 \end{cases}$.

The distribution mapping is defined by

$DM(l_1((d,0) + (b,0) + (u,0)) + l_2(v,0) + l_3((d,0) + (e,0) + (v,0)) + l_4(u,0)) =$

$= w' + l_1b + (l_1 + l_3)d + l_3e + (l_1 + l_4)u + (l_2 + l_3)v,$

where w' is the content of B before applying the distribution mapping.

Proposition 3. *$\mathcal{M}$ can compute the product of two positive integers*

Proof. We insert initially $na + mb + u = na + mb + 0d + 0e + 1u + 0v$ in the resource box; the result is obtained as the coefficient of d. We have the following sequence of computations.

Step 1
We consume first one a and one u to produce one v:
$na + mb + u = na + mb + 0d + 0e + 1u + 0v \Longrightarrow_{CRM}$
$(0, a + u, 0, 0, (n-1)a + mb) \Longrightarrow_{\bigwedge \mathcal{A}_i}$
$(0, (v, 0), 0, 0, (n-1)a + mb) \Longrightarrow_{DM} (n-1)a + mb + v.$
Next m steps consume m objects b, and produce m objects d and m objects e:
$\Longrightarrow_{CRM} (0, 0, b + v, 0, (n-1)a + (m-1)b) \Longrightarrow_{\bigwedge \mathcal{A}_i}$
$(0, 0, (d, 0) + (e, 0) + (v, 0), (n-1)a + (m-1)b) \Longrightarrow_{DM}$
$(n-1)a + (m-1)b + d + e + v \Longrightarrow_{CRM}$
$(0, 0, b + v, 0, (n-1)a + (m-2)b + d + e) \Longrightarrow_{\bigwedge \mathcal{A}_i}$
$(0, 0, (d, 0) + (e, 0) + (v, 0), (n-1)a + (m-2)b + d + e) \Longrightarrow_{DM}$
$(n-1)a + (m-2)b + 2d + 2e + v$
$\ldots$
$\Longrightarrow_{CRM} (0, 0, b + v, 0, (n-1)a + (m-1)d + (m-1)e) \Longrightarrow_{\bigwedge \mathcal{A}_i}$
$(0, 0, (d, 0) + (e, 0) + (v, 0), (n-1)a + (m-1)d + (m-1)e) \Longrightarrow_{DM}$
$(n-1)a + md + me + v.$
Step 2
We consume one a and one v in order to produce one u:
$(n-1)a + md + me + v \Longrightarrow_{CRM} (0, 0, 0, a + v, (n-2)a + md + me) \Longrightarrow_{\bigwedge \mathcal{A}_i}$
$(0, 0, 0, (u, 0), (n-2)a + md + me) \Longrightarrow_{DM} (n-2)a + md + me + u$
Next m steps consume m objects e, and produce m objects b and m objects d:
$\Longrightarrow_{CRM} (e + u, 0, 0, 0, (n-2)a + md + (m-1)e) \Longrightarrow_{\bigwedge \mathcal{A}_i}$
$((d, 0) + (b, 0) + (u, 0), 0, 0, 0, (n-2)a + md + (m-1)e) \Longrightarrow_{DM}$
$(n-2)a + b + (m+1)d + (m-1)e + u \Longrightarrow_{CRM}$
$((d, 0) + (b, 0) + (u, 0), 0, 0, 0, (n-2)a + b + (m+1)d + (m-2)e) \Longrightarrow_{DM}$
$(n-2)a + 2b + (m+2)d + (m-2)e + u$
$\ldots$
$\Longrightarrow_{CRM} (e + u, 0, 0, 0, (n-2)a + (m-1)b + (2m-1)d) \Longrightarrow_{\bigwedge \mathcal{A}_i}$
$((d, 0) + (b, 0) + (u, 0), 0, 0, 0, (n-2)a + (m-1)b + (2m-1)d) \Longrightarrow_{DM}$
$(n-2)a + mb + 2md + u$

Return to Step 1
The computations goes on until the all the objects a are consumed from B.

Both steps have $(m + 1)$ computations steps defined by the application of CRM followed by the application of the restricted direct product of MmA's and an application of DM. Hence after $2(m + 1)$ computation steps, we consume $2a$ in order to produce $2md$. Finally we obtain the following multisets in the box:

1. If $n = 2k + 1$ is an odd integer, then we execute k pairs (**Step 1, Step 2**), then one additional **Step 1**, and finally the box contains $mnd + me + v$;
2. If $n = 2k$ is an even integer, then we execute k pairs (**Step 1, Step 2**), and the box contains $mb + mnd + u$.

In both situations we get mn objects d. □

Remark 1. An interesting feature of the membrane systems for multiplication presented in this paper is that the computation may continue after reaching a

certain result, and so the system acts as a P transducer [6]. Thus if initially there are n (objects a) and m (objects b), the system evolves and produces $n \cdot m$ objects d. Afterwards, the user can inject more objects a and the system continues the computation obtaining the same result as if the objects a are present from the beginning. For example, if the user wishes to compute $(n + k) \cdot m$, it is enough to inject k objects a at any point of the computation.

5 Connecting Simple P Machines

Let us to consider a family of simple P machines indexed by a finite set T (of targets). We can define the *neighborhood* of a simple P machine $\mathcal{M}^j$ ($j \in T$) to be the set of all the simple P machines which can communicate with $\mathcal{M}^j$. In a P system environment (i.e., hierarchical system of simple P machines), by a neighborhood of $\mathcal{M}^j$ we understand its parent, its children, and itself.

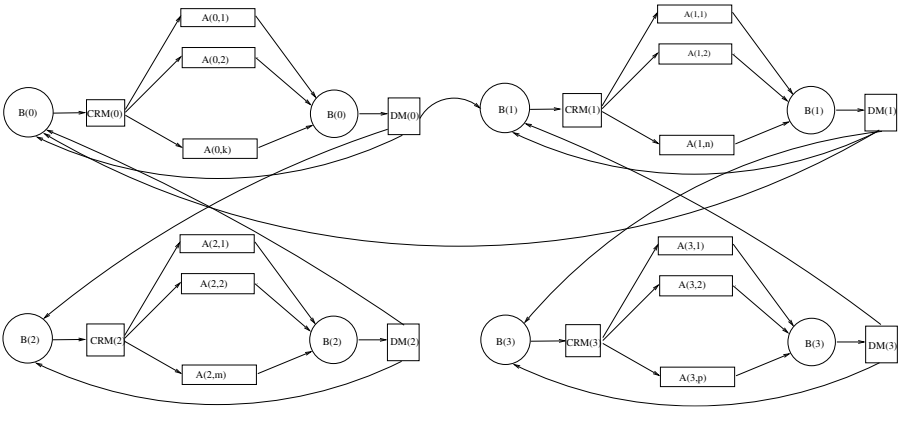
The output of a simple P machine $\mathcal{M}^j = (V, \bigwedge_{i=1}^{n_j} \mathcal{A}_i^j, O, B^j, CRM^j)$ is given by a multiset from $\mathbb{N}\langle V \times T\rangle$ of the form $w''^j + \sum_{i=1}^{n_j} k_i \cdot (y_i, tar_i)$; it can be viewed as a translation mapping from $\mathbb{N}\langle V\rangle$ into $\mathbb{N}\langle V \times T\rangle$. w''^j represents the updated content of the box; it contains the unprocessed multiset w'^j added with possible other multisets from its neighborhood.

We show how we can connect a simple P machine with other simple P machines. This can be done by using a cascade-like product and the *distribution mapping* $DM^j : \mathbb{N}\langle V \times T\rangle \to (\mathbb{N}\langle V\rangle)^{m_j}$, where m_j is the number of simple P machines in the neighborhood of $\mathcal{M}^j$.

$$DM^j(\sum_{i=1}^{m_j} k_i \cdot (y_i, tar_i)) = (w^1 + k_1 \cdot y_1, w^2 + k_2 \cdot y_2, \ldots, w^{m_j} + k_{m_j} \cdot y_{m_j}),$$

where w^i represents the box content of the simple P machine indexed by i.

An example of a graphical representation of a network composed of four simple P machines $\mathcal{M}^0$, $\mathcal{M}^1$, $\mathcal{M}^2$ and $\mathcal{M}^3$ is given by the following picture:



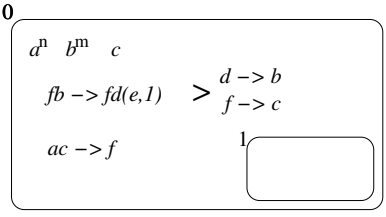


Fig. 3. P system with two membranes

The simple P machine $\mathcal{M}^0$ (left upper corner) is the skin membrane of this network. We have two children represented by the simple P machines $\mathcal{M}^1$ (right upper corner) and $\mathcal{M}^2$ (left lower corner), and the simple P machine $\mathcal{M}^3$ (right lower corner) is the child of $\mathcal{M}^1$. As far as we can observe, $\mathcal{M}^3$ can communicate only with $\mathcal{M}^1$, $\mathcal{M}^1$ can communicate only with $\mathcal{M}^0$ and $\mathcal{M}^3$, and $\mathcal{M}^2$ can communicate only with $\mathcal{M}^0$. If we denote by $N(j)$ the set of indexes of the simple P Machines in the neighborhood of the sPM indexed by j, in our diagram we have following sets: $N(0) = \{0, 1, 2\}$, $N(1) = \{0, 1, 3\}$, $N(2) = \{0, 2\}$, and $N(3) = \{1, 3\}$.

Figure 3 presents a P system with two membranes which also computes the multiplication of two numbers n and m. The P system (with two membranes) is given by

$$\Pi = (\{a, b, c, d, e, f\}, \{e\}, \emptyset, [_0[_1]_1]_0, a^n b^m c, \emptyset, (R_0, \rho_0), (\emptyset, \emptyset), 1), \text{ where}$$

- $R_0 = \{r_0 : ac \rightarrow f,\ r_1 : fb \rightarrow fd(e,1),\ r_2 : f \rightarrow c,\ r_3 : d \rightarrow b\}$
- $\rho_0 = \{r_1 > r_2,\ r_1 > r_3\}$

We consider two simple P machines $\mathcal{M}^0 = (V, \bigwedge_{i=1}^4 \mathcal{A}_i^0, O, B^0, CRM^0)$, and $\mathcal{M}^1 = (V, \emptyset, O, B^1, \emptyset)$ where the alphabets are $V = \{a, b, c, d, e, f\}$ and $O = \{(a,0), (b,0), (c,0), (d,0), (e,0), (f,0), (a,1), (b,1), (c,1), (d,1), (e,1), (f,1)\}$
$\mathcal{A}_1^0$ translates $a + c$ in $(f, 0)$, $\mathcal{A}_2^0$ translates $f + b$ in $(f,0) + (d,0) + (e,1)$, $\mathcal{A}_3^0$ translates f in $(c, 0)$, and $\mathcal{A}_4^0$ translates d in $(b, 0)$.
CRM^0 is defined as

$CRM^0(k_1a + k_2b + k_3c + k_4d + k_5e + k_6f) = (l_1(a + c), l_2(f + b), l_3f, l_4d, w')$, where

$l_1 = min\{k_1, k_2\}, l_2 = min\{k_2, k_6\}, l_3 = \begin{cases} 0 & l_2 \neq 0 \\ k_6 & l_2 = 0 \end{cases}, l_4 = \begin{cases} 0 & l_2 \neq 0 \\ k_4 & l_2 = 0 \end{cases}$

The distribution mapping is defined by

$DM(l_1(f,0) + l_2((f,0) + (d,0) + (e,1)) + l_3(d,0) + l_4(b,0)) =$

$(w' + (l_1 + l_2)f + (l_2 + l_3)d + l_4b, w" + l_2e)$, where w' and $w"$ represent the contents of B^0 and B^1, respectively, before applying distribution.

It can be verified that if we insert initially $na + mb + c$ in B^1, after doing the computation, we get $c + md$ in B^1 and mne in B^2.

6 Conclusion and Further Work

According to our knowledge, we present for the first time an automaton corresponding faithfully to a membrane system with a single membrane and emphasizing on its maximal parallel evolution rules. We call simple P machine such an automaton. We give examples of simple P machines for both P systems with promoters and P systems with priorities. For each case we show that the P machines provide the same result as their corresponding P systems. We present a way of connecting simple P machines according to their communication rules, and give am example how the new resulting network corresponds to P systems with more than one membrane.

Considering the class $\mathcal{P}$ of the P systems involving priorities, promoters, activators, or catalysts, we claim that we can build a P machine having the same behavior and result. In such a P machine

- every rule is modeled by a very simple Mealy multiset automata;
- various features involved by priority, promoters, as well as maximal parallel and nondeterministic application of the rules are captured by the control resource mappings;
- communications are modeled by the distribution mappings.

The converse of this claim is true because of the universality of P systems. However it is hard to have a constructive converse, namely an effective procedure of building a P systems having the same behavior and results as a given P machine. In fact we use very simple MmA's incorporated in sPM to model a P system; they have only one generator (i.e., the left-hand side of the rule) for the set of tc-multisets (a cyclic semimodule). On the other hand, the definition of a P machine does not restrict the Mealy multiset automata representing its parallel components to have only one generator for the set of all tc-multiset. This means that such a machine can change the "rules" after every computation step, for instance. These things are subject of further investigations.

Another direction of further research is given by the definition and the study of various aspects of machine-like theory (behavior, bisimulation, composition). It is also interesting to continue the study of Mealy multiset automata by using linear algebra, detecting, for example, which are the links between the matrix associated with two different tc-multisets.

Automata theory has mainly a sequential nature, in contrast with P systems. Membrane systems represent abstract models inspired by the compartments of a cell. We try to connect the theory of membrane computing with the classic theory of (Mealy) automata. On the other hand, our machine has the capability to be highly adaptable, i.e., we can easily pass from strings to multisets and back, and so on. The inductive description is not able to distinguish between deterministic and nondeterministic automata. As we are more interested in their behavior, we think to a co-inductive point of view (see [5]). It is worth to point out that while strings are of algebraic nature, multisets can be also viewed as their duals, so they have a coalgebraic nature.

References

1. F. Bernardini, V. Manca. Dynamical aspects of P systems. *BioSystems*, 70 (2002), 85–93.
2. C. Bonchiş, G. Ciobanu, C. Izbaşa. Encodings and arithmetic operations in membrane computing. In *Theory and Applications of Models of Computation*, LNCS 3959, Springer, 2006, 618–627.
3. L. Cardelli. Languages and notations for systems biology. *Unconventional Programming Paradigms*, Le Mount St.Michel, 2004.
4. G. Ciobanu, M. Gontineac. Mealy multiset automata. *International Journal of Foundations of Computer Science*, 17 (2006), 111–126.
5. G. Ciobanu, M. Gontineac. Algebraic and coalgebraic aspects of membrane computing. In *Membrane Computing. WMC6*, LNCS 3850, Springer, 2006, 181–198.
6. G. Ciobanu, Gh. Păun, Gh. Ştefănescu. P transducers, *New Generation Computing*, 24 (2006), 1–28.
7. E. Csuhaj-Varju, C. Martin-Vide, V. Mitrana. Multiset automata. In *Multiset Processing: Mathematical, Computer Science, and Molecular Computing Points of View*, LNCS 2235, Springer, 2001, 69–83.
8. E. Csuhaj-Varju, G. Vaszil. P automata or purely communicating accepting P systems. In *Membrane Computing. WMC-CdeA 2002*, LNCS 2597, Springer, 2003, 219–233.
9. M. Oswald. *P Automata*, PhD Thesis, Technical University Vienna, 2004.
10. Gh. Păun. *Membrane Computing. An Introduction*. Springer, 2002.

Modeling Dynamical Parallelism in Bio-systems

Erzsébet Csuhaj-Varjú[1], Rudolf Freund[2], and Dragoş Sburlan[3,4]

[1] Computer and Automation Research Institute
Hungarian Academy of Sciences
Kende utca 13–17, H-1111 Budapest, Hungary
csuhaj@sztaki.hu
[2] Faculty of Informatics
Vienna University of Technology
Favoritenstr. 9–11, A–1040 Vienna, Austria
rudi@emcc.at
[3] Department of Computer Science and Artificial Intelligence
University of Seville,
Av. Reina Mercedes, 41012, Seville, Spain
[4] Faculty of Mathematics and Informatics
Ovidius University of Constantza,
124 Mamaia Bd., Constantza, Romania
dsburlan@univ-ovidius.ro

Abstract. Among the many events that occur in the life of biological organisms there are multitudes of specific chemical transformations that provide the cell with usable energy and molecules needed to form its structure and coordinate its activities. These biochemical reactions, as well as all other cellular processes, are governed by basic principles of chemistry and physics. A significant factor that determines whether or not reactions could take place is the entropy (it measures the randomness of the system). This measure depends on various factors. In an abstract framework, all these factors, which describe the way molecules interact, can be expressed by means of a computable multi-valued function that, depending on the current state of the system, establishes the possible ways of the evolution of the system. Inspired by these facts, we introduce and study several bio-mimetic computational rewriting systems that use discrete components (i.e., finite alphabets, finite set(s) of rewriting rules, etc.) and perform their computational steps in a non-deterministic manner and in a degree of rewriting parallelism that depends on the current state of the system, both specified by a given multi-valued function. Furthermore, we describe systems which produce the same output independently of the values taken by the considered functions.

1 Introduction

In nature, we often find biological systems that are not necessarily homogeneous, but consist of many discrete, interacting entities that have a certain physical spatial distribution. This fact suggests that even if these entities interact in a parallel manner, they obey to some local conditions (concentration, for instance)

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 330–351, 2006.

and therefore the interaction parallelism cannot be considered as maximal or fixed, but as a variable that depends on the state of the system.

For example, at the cell level, bio-molecular mechanisms are the result of many different chemical reactions that take place with a certain degree of mutual independence but in such way that they finally (and amazingly) exhibit an overall co-ordination. Traditionally, these behaviors were modeled by using the theory of partial derivative equations and non-linear dynamical systems. However, this approach usually gave the general evolution and the dynamics of the system but not always the exact solution. From a discrete point of view, the interest was mainly in inferring the properties of the languages generated by such bio-inspired models. Although discrete models of complex phenomena may generate errors, the magnitude of the errors can be reduced arbitrarily by considering a better granularity of the phenomenon. This approach in general leads to a high computational effort, so a more efficient way of studying properties of bio-systems might be to consider discrete formal systems that, in their formal description have embedded a certain degree of randomness, describing the way systems evolve.

One such property regards the measure of parallelism. From this point of view, for instance, using biochemical reasoning, one might predict that, given a particular state of a bio-system and the rules that make it evolve, an approximate next state is reached after a certain time. Basically, even if one does not know the exact number of times the rules are applied, one knows that after a particular time the reactions that had the potential to be applied, were actually accomplished in an approximate rate with respect to the state of the system.

Moreover, from the point of view of computer science, in case we are trying to make use of bio-systems as computational devices we should be able to control their behavior irrespectively of the rate of parallelism occurring within them. Therefore, we are interested in systems that one might call "parallel fault tolerant" which means that they produce the same output no matter which is the "evolution" of the parallelism. This assumption might also have a biological counterpart, namely, natural sub-systems are able to regulate themselves and replace, in case is needed, the functions of other sub-systems such that the overall system can perform the same task. From this point of view, one can assume that a complex bio-system (like a cell or whatever organism) has the ability to reach a "desired" state, no matter how "local" decisions were made.

Here we will consider two bio-mimetic models, namely Lindenmayer systems, inspired by the development of multi-cellular organisms, and P systems with promoters, motivated by enzyme activation/inactivation taking place in the living cells.

We will extend the original definitions of these systems by considering computable multi-valued functions that control the derivation (in terms of specifying the rewriting parallelism and the nondeterminism).

2 Preliminaries

We assume the reader to be familiar with basic notions of formal languages, in particular Lindenmayer systems, and P systems (for more details one can consult [4], [5], and [6]).

We start by briefly recalling some basic notions concerning Lindenmayer systems and then we present some elementary notions concerning multisets. Later on, we will introduce several new definitions needed for the purpose of this work.

A $0L$ system (an interactionless L-system) is a triple $H = (V, R, w)$, where V is an alphabet, $w \in V^+$, the axiom, and R is a set of rules (productions) of the form $a \to v$, where $a \in V$ and $v \in V^*$. Moreover, the production set R is complete: for every $a \in V$ there is a rule of the form $a \to v$, $v \in V^*$ in R. The direct derivation relation in a $0L$ system $H = (V, R, w)$ is defined as follows: for $x, y \in V^*$ we write $x \Longrightarrow_R y$ if $x = a_1 \dots a_n$, $y = z_1 z_2 \dots z_n$, $a_i \in V$, $z_i \in V^*$, $1 \leq i \leq n$, and $a_i \to z_i \in R$. We denote the reflexive and transitive closure of $\Longrightarrow_R$ by $\Longrightarrow_R^*$.

A $T0L$ system (a tabled $0L$ system) with k tables, $k \geq 1$, is a construct $H = (V, T, w)$ where $T = \{T_1, \dots, T_k\}$ and each triple (V, T_i, w), $1 \leq i \leq k$, is a $0L$ system. A string x directly derives a string y in H, $x, y \in V^*$, if and only if y is directly generated from x by applying some of the tables of H, say, T_i.

A $T0L$ system, where a subalphabet Δ of the alphabet V is distinguished as the terminal alphabet, written as $H = (V, T, w, \Delta)$, is called an $ET0L$ system (an extended $T0L$ system) and its language is defined by $L(H) = \{v \in T^* \mid w \Longrightarrow^* v\}$.

We now turn to the basic notions concerning multisets. A *multiset* over an arbitrary set X is a mapping $M : X \longrightarrow I\!N$. By $M(x)$, $x \in X$, we denote the multiplicity of x in the multiset M. If the set $X = \{x_1, \dots, x_n\}$ is finite, then the multiset M can explicitly be given in the form $\{(x_1, M(x_1)), \dots, (x_n, M(x_n))\}$. The support of a multiset M is the set $supp(M) = \{x \in X \mid M(x) \geq 1\}$. A multiset M is empty if its support is empty. Let $M_1, M_2 : X \longrightarrow I\!N$ be two multisets. We say that M_1 is included in M_2 (denoted by $M_1 \subseteq M_2$) if $M_1(x) \leq M_2(x)$, for all $x \in X$. The inclusion is strict if $M_1 \subseteq M_2$ and $M_1 \neq M_2$. The union (difference) of two multisets, $M_1 \cup M_2 : X \longrightarrow I\!N$ (respectively, $M_1 \setminus M_2 : X \longrightarrow I\!N$), is defined as $(M_1 \cup M_2)(x) = M_1(x) + M_2(x)$ (respectively, for $M_2 \subseteq M_1$, $(M_1 \setminus M_2)(x) = M_1(x) - M_2(x)$), for all $x \in X$. A multiset M of finite support, $\{(x_1, M(x_1)), \dots, (x_n, M(x_n))\}$, can also be represented by the string $w = x_1^{M(x_1)} x_2^{M(x_2)} \dots x_n^{M(x_n)}$, and all the permutations of this string precisely identify the objects in the support of M and their multiplicities in M. We note that the Parikh image of w, $\Psi_X(w)$, is exactly the vector $(M(x_1), \dots, M(x_n))$ of the multiplicities. The cardinality of a multiset $w = x_1^{M(x_1)} x_2^{M(x_2)} \dots x_n^{M(x_n)}$ is $card(w) = M(x_1) + M(x_2) + \dots + M(x_n)$; the number of occurrences of x_i in w is denoted by $|w|_{x_i} = M(x_i)$, for $1 \leq i \leq n$.

Let $l \in I\!N$ and let $w = x_1^{t_1} \dots x_n^{t_n}$, $x_i \in X$, $t_i \in I\!N$, $1 \leq i \leq n$, be a multiset over X. We define the product $l * w = x_1^{l \cdot t_1} x_2^{l \cdot t_2} \dots x_n^{l \cdot t_n}$.

Let us consider a finite set of symbols $V = \{a_1, a_2, \ldots, a_n\}$. A multiset rewriting rule is a pair (u, v) where $u \in V^+$, $v \in V^*$ represent multisets over the set V; such a rule can also be written as $u \to v$. For a multiset rewriting rule $r : u \to v$, with $u, v \in V^*$ being multisets over V, let $left(r) = u$ and $right(r) = v$.

In what follows we formally define the conditions required for a given multiset rewriting rule (or several multiset rewriting rules) to be applied (to be applied simultaneously, respectively) on a given multiset of symbols.

Let $w \in V^*$ be a multiset over V and let $R = \{r_1, r_2, \ldots, r_k\}$ be a set of multiset rewriting rules such that $r_i = u_i \to v_i$, with $u_i, v_i \in V^*$, $1 \leq i \leq k$. Let us denote the set of *applicable* multiset rewriting rules to w by $R_w^{ap} \subseteq R$, i.e., $R_w^{ap} = \{r \in R \mid left(r) \subseteq w\}$.

By $R_w^{sap} = r_1^{t_1} r_2^{t_2} \ldots r_k^{t_k}$, $t_i \in I\!N$, $1 \leq i \leq k$, we denote the multiset over R of multiset rewriting rules being *simultaneously applicable* to w. R_w^{sap} is any multiset such that

$$\bigcup_{1 \leq i \leq k} t_i * left(r_i) \subseteq w. \tag{1}$$

By R_w^{SAP} we denote the set of all multisets of rules simultaneously applicable to w, i.e., $R_w^{SAP} = \{R_w^{sap} \text{ satisfying } (1) \mid R_w^{sap} \in R^*\}$.

Next, we define the "impact" of a multiset of rules when they are applied to a given multiset of symbols (i.e., how many distinct symbols are rewritten). Based on this concept, we further define the set containing the multisets of rules that produce the largest "impact" on a given multiset of symbols.

For $x = r_1^{i_1} r_2^{i_2} \ldots r_k^{i_k} \in R_w^{SAP}$, let

$$D_x = supp(\bigcup_{i=1}^{k} left(r_i)).$$

The set D_x indicates all *distinct* symbols from w that are rewritten by an application of x.

By

$$R_w^{MSAP} = \{x = r_1^{i_1} r_2^{i_2} \ldots r_k^{i_k} \in R_w^{SAP} \mid card(D_x) = \max_{y \in R_w^{SAP}} (card(D_y))\}$$

we denote the set of multisets of rules simultaneously applicable to w, called the *maximal component* of R_w^{SAP}.

Remark 1. The maximal component of R_w^{SAP} contains all multisets of rules simultaneously applicable to w such that the rewriting of distinct symbols is maximal (in the sense of the processed objects).

Let Y be a set of multisets over R; we denote

$$Pr(Y) = \{r_1 r_2 \ldots r_k \mid r_1^{t_1} r_2^{t_2} \ldots r_k^{t_k} \in Y\}.$$

We will use the set of sets $W_w = \{X \subseteq R_w^{MSAP} \mid Pr(X) = Pr(R_w^{MSAP})\}$.

Let

$$R_w^{MAX} = \{x \in R_w^{SAP} \mid \text{there exists a} y \in R_w^{SAP} \text{ such that } x \subseteq y \text{ implies } x = y\}$$

be *the set of multisets of all maximal rules simultaneously applicable* to w.

Remark 2. The concept of maximal parallelism of rewriting (used, for instance, in the P systems framework) is expressed using the set $R_w^{MAX} \subseteq R_w^{SAP}$. For example, considering a P system Π in a given configuration and a region of Π containing a set of rules R that acts on the multiset w, an element in R_w^{MAX} gives a possible ensemble of rules that can be applied on w in a maximally parallel manner.

In addition, one can remark that $R_w^{MSAP} \supseteq R_w^{MAX}$. Observe that for non-cooperative multiset rewriting rules $Pr(R_w^{MAX}) = Pr(R_w^{MSAP})$.

The notions presented above regarding multisets and rules can be extended to strings and productions in a straightforward manner as follows[1]. However, as opposed to the multiset case, here we have to pay more attention to the implicit order of the symbols in a string.

Considering a set of productions $R = \{r_1, r_2, \ldots, r_k\}$ over an alphabet V, the set of productions *applicable* to a string w is $RS_w^{ap} = \{r \in R \mid w = \alpha_1 left(r)\alpha_2, \alpha_1, \alpha_2 \in V^*\}$. Let $U = \{\overline{a} \mid a \in V\}$ and let $o : V^* \to U^*$ be a morphism that maps symbols from V into their corresponding overlined symbols. Let $h_{r_i} : (V \cup U)^* \to (V \cup U)^*$ such that $h_{r_i}(\alpha_1 left(r_i)\alpha_2) = \alpha_1 o(left(r_i))\alpha_2$, $\alpha_1, \alpha_2 \in (V \cup U)^*$. Then we can define a multiset of productions *simultaneously applicable* to a string w as follows (recall that we are not interested which are the sites in w where the productions from R are applied, but only if they can be applied simultaneously). $RS_w^{sap} = r_1^{t_1} \ldots r_k^{t_k}$ such that there exists $h_{r_1}^{t_1}(h_{r_2}^{t_2}(\ldots h_{r_k}^{t_k}(w)\ldots))$, i.e., there exist "enough" sites in w where the productions could be applied. The set of all multisets of productions simultaneously applicable to the string w is denoted by RS_w^{SAP}.

The set of all *distinct* symbols from the string w rewritten by an application of the multiset of productions $x = r_1^{i_1} r_2^{i_2} \ldots r_k^{i_k} \in RS_w^{SAP}$ is

$$DS_x = \{a \in V \mid (\exists)\, r_i, 1 \leq i \leq k, \text{ such that } left(r_i) = \alpha_1 a \alpha_2, \alpha_1, \alpha_2 \in V^*\}.$$

The *maximal component* of RS_w^{SAP} is defined as for multisets, i.e.,

$$RS_w^{MSAP} = \{x = r_1^{i_1} r_2^{i_2} \ldots r_k^{i_k} \in RS_w^{SAP} \mid card(DS_x) = \max_{y \in RS_w^{SAP}}(card(DS_y))\}.$$

Let Y be a set of multisets over R; we denote

$$Pr(Y) = \{r_1 r_2 \ldots r_k \mid r_1^{t_1} r_2^{t_2} \ldots r_k^{t_k} \in Y\}.$$

We will use the set of sets $WS_w = \{X \subseteq RS_w^{MSAP} \mid Pr(X) = Pr(RS_w^{MSAP})\}$.

[1] We will use similar notations as for the multiset case, the difference being a capital letter S on the right-hand side of each defined operator.

Example 1. Let $V = \{a, b, c\}$. Consider the multiset $w = aaabbbbccc$ and the set of multiset rewriting rules $R = \{r_1 : abc \to \alpha, r_2 : bcc \to \beta, r_3 : aac \to \gamma, r_4 : acccc \to \theta\}$. Then we have:

- $R_w^{SAP} = \{r_1, r_1^2, r_1^3, r_2, r_3, r_1r_2, r_1r_3, r_2r_3\}$;
- $R_w^{MAX} = \{r_1^3, r_1r_2, r_1r_3, r_2r_3\}$;
- $D_{r_1r_3} = D_{r_1} = \{a, b, c\}$; $D_{r_3} = D_{r_4} = \{a, c\}$;
- $R_w^{MSAP} = \{r_1, r_1^2, r_1^3, r_1r_2, r_1r_3, r_2r_3\}$.

Example 2. Let $V = \{a, b, c, d\}$. Consider the string $w = aabc$ and the set of rewriting productions $R = \{r_1 : a \to \alpha_1, r_2 : a \to \alpha_2, r_3 : b \to \beta, r_4 : c \to \gamma, r_5 : d \to \theta\}$. Then we have:

$$\begin{aligned}
RS_w^{SAP} &= \{r_1, r_2, r_1^2, r_2^2, r_3, r_4, r_1r_2, r_1r_3, r_2r_3, r_1r_4, r_2r_4, r_3r_4\} \\
&\cup \{r_1^2r_3, r_2^2r_3, r_1^2r_4, r_2^2r_4, r_1r_2r_3, r_1r_3r_4, r_2r_3r_4\} \\
&\cup \{r_1^2r_3r_4, r_2^2r_3r_4, r_1r_2r_3r_4\}; \\
RS_w^{MAX} &= \{r_1^2r_3r_4, r_2^2r_3r_4, r_1r_2r_3r_4\}; \\
RS_w^{MSAP} &= \{r_1r_3r_4, r_2r_3r_4, r_1^2r_3r_4, r_2^2r_3r_4, r_1r_2r_3r_4\} \\
WS_w &= \{\{r_1r_3r_4, r_2^2r_3r_4, r_1r_2r_3r_4\}, \{r_1r_3r_4, r_2r_3r_4, r_1r_2r_3r_4\}, \\
&\quad \{r_1^2r_3r_4, r_2^2r_3r_4, r_1r_2r_3r_4\}, \ldots\}.
\end{aligned}$$

3 On Dynamical Parallelism in L Systems

In this section we extend the classical definition of Lindenmayer system in order to fit a more general perspective. The new constructs model developmental systems in which parts of the organism change simultaneously but not necessarily in the totally parallel manner as in the case of the classical Lindenmayer systems, but determined by the current state of the organism. We present several results regarding the computational power of these systems.

Computing formal systems, usually, make use of rewriting rules to perform their computations. The semantics of such formal models provides the ways how the rewriting rules are applied. Here, in order to capture the most general case, we will consider computable multi-valued functions that, depending on the current state of the system, control the applications of the rules. This assertion can be better understood if we provide the reader with some biological motivation.

For a given state of a bio-system (represented by a multiset/string w), one can predict that a certain rule, say $a \to \alpha$, is to be applied on w in a rate specified by a value in the interval $(x, y) \subseteq (0, 1)$. Therefore, in that computational step, the rule $a \to \alpha$ is applied a number of times i, such that $\lceil x \cdot |w|_a \rceil \leq i \leq \lceil y \cdot |w|_a \rceil$. However, when generalizing this concept, we might assume that there is more than one interval that control the applications of the rules, and this brings us to the following formalism.

Definition 1. *An M-rate $0L$ system, or an $M0L$ system, is a quadruple $H = (V, R, \omega, f)$, where:*

- $V = \{a_1, \ldots, a_m\}$ *is a finite alphabet,*
- R *is a finite set of rules of the form* $a \to \alpha$, *where* $a \in V$ *and* $\alpha \in V^*$. *Moreover,* R *is complete, i.e., for each symbol* $a \in V$ *there exists at least one production* $a \to \alpha \in R$ *with* $\alpha \in V^*$. *The productions in* R *are uniquely labeled, i.e., we associate to each production* $a \to \alpha$, *where* $a \in V$ *and* $\alpha \in V^*$, *a unique label* l *and then we also can write the production in the form* $l : a \to \alpha$.
- f *is a multi-valued computable function such that* $f : V^* \to \mathscr{P}(R^*)$ *with* $f(x) \in \mathscr{P}(RS_x^{SAP})$ *for* $x \in V^*$,
- $\omega \in V^*$ *is the axiom.*

$M0L$ systems use M-rate parallel derivations, i.e., $x \in V^*$ directly derives $y \in V^*$ in a $M0L$ system $H = (V, R, \omega, f)$, written as $x \underset{f}{\Longrightarrow} y$, if $x = x_1 x_2 \ldots x_n$, $y = y_1 y_2 \ldots y_n$, $x_i \in V$, $y_i \in V^*$, $1 \leq i \leq n$, and the following conditions hold:

- for every j, $1 \leq j \leq n$, either $y_j = x_j$ (the j-th symbol remains unchanged), or $r : x_j \to y_j \in P$ (some production of R is applied to the j-th symbol);
- the multiset of productions applied to x is in $f(x)$, where $f(x) \in \mathscr{P}(RS_x^{SAP})$. (Thus, if the multiset of the productions applied to x is $r_1^{t_1} r_2^{t_2} \ldots r_k^{t_k} \in f(x)$, then production r_i is applied t_i times to x, for $1 \leq i \leq k$).

This manner of derivation is called the *"weak mode"*. In the case of the *"strong mode"* of derivation, we consider $f : V^* \to \mathscr{P}(R^*)$, $f(x) \in \{X \subseteq RS_x^{MSAP} \mid Pr(X) = Pr(RS_x^{MSAP})\}$. (Thus, if a symbol appears in the string x, then at least one occurrence of it is rewritten.)

The transitive and reflexive closure of $\overset{M0L}{\Longrightarrow}_H$ is denoted by $\overset{M0L}{\Longrightarrow}^*_H$. The generated language of the $M0L$ system H is

$$L(H) = \{u \in V^* \mid \omega \overset{M0L}{\Longrightarrow}^*_H u\}.$$

Remark 3. For both derivation modes, the *local degree of parallelism* for a given derivation step is defined by

$$\max_{y \in f(x)} (card(supp(y))).$$

For the weak mode of derivation, a degree of parallelism k, $1 \leq k \leq 2$, means that at most two distinct productions are applied simultaneously. The computable multi-valued function f defines how many times a certain production is applied to a given string.

Definition 2. *An* M*-rate* $T0L$ *system, or an* $MT0L$ *system, is a triplet* $H = (V, T, \omega)$, *where:*
- V *is a finite alphabet,*
- $T = \{(T_1, f_1), \ldots, (T_k, f_k)\}$ *is a finite set of pairs, where each* T_i, $1 \leq i \leq k$, *is a complete set of productions of the form* $a \to \alpha$, *where* $a \in V$, $\alpha \in V^*$, *and each* f_i, $1 \leq i \leq k$, *is a computable multi-valued function such that* $f_i : V^* \to \mathscr{P}(T_i^*)$ *with* $f_i(x) \in \mathscr{P}(T_i S_x^{SAP})$ *for* $x \in V^*$,
- $\omega \in V^*$ *is the axiom.*

As for $M0L$ systems, this way of defining f_i, $1 \leq i \leq k$, stands for the weak mode of derivation and for the strong mode of derivation, the multi-valued functions are defined as $f_i : V^* \to \mathscr{P}(T_i^*)$ with $f_i(x) \in \{X \subseteq T_i S_x^{MSAP} \mid Pr(X) = Pr(T_i S_x^{MSAP})\}$ for $x \in V^*$, $1 \leq i \leq k$.

We say that x directly derives y in an $MT0L$ system $H = (V, T, \omega)$, with $x, y \in V^*$, written as $x \overset{MT0L}{\Longrightarrow}_H y$, if $x \overset{M0L}{\Longrightarrow}_{H_i} y$ for some i, $1 \leq i \leq k$, with the $M0L$ system $H_i = (V, T_i, \omega, f_i)$.

The transitive and reflexive closure of $\overset{MT0L}{\Longrightarrow}_H$ is denoted by $\overset{MT0L}{\Longrightarrow}^*_H$. The generated language of the $MT0L$ system H is

$$L(H) = \{u \in V^* \mid \omega \overset{MT0L}{\Longrightarrow}^*_H u\}.$$

Definition 3. *An M-rate $ET0L$ system, or an $MET0L$ system, is a quadruple $H = (V, T, \omega, \Delta)$, where $\overline{H} = (V, T, \omega)$ is an $MT0L$ system, and $\Delta \subseteq V$, $\Delta \neq \emptyset$, is the terminal alphabet. In an $MET0L$ system $H = (V, T, \omega, \Delta)$, x directly derives y, for $x, y \in V^*$, written as $x \overset{MET0L}{\Longrightarrow}_H y$, if $x \overset{MT0L}{\Longrightarrow}_{\overline{H}} y$.*

The transitive and reflexive closure of $\overset{MET0L}{\Longrightarrow}_H$ is denoted by $\overset{MET0L}{\Longrightarrow}^*_H$.

The language generated by the $MET0L$ system H is $L(H) = \{w \in \Delta^* \mid \omega \overset{MET0L}{\Longrightarrow}^*_H w\}$.

Definition 4. *An $M0L$ ($MT0L$, $MET0L$) system generating the same language independently of the functions associated with the set(s) of productions is called* parallel-free *$M0L$ ($MT0L$, $MET0L$) system.*

The families of languages generated by M0L (or MT0L, MET0L) systems working in the strong mode are denoted by $M0L^s$ (or $MT0L^s$, $MET0L^s$, respectively).

The families of languages generated by M0L (or MT0L, MET0L, respectively) systems working in the weak mode are denoted by $M0L^w$ (or $MT0L^w$, $MET0L^w$, respectively).

When we speak about the families of languages mentioned above, we will denote the parallel-free property by adding the superscript pf.

$U = \{L \mid card(L) = 1\}$ denotes the family of all singleton languages.

The following result describes the computational power of extended, interactionless M-rate Lindenmayer systems working in the weak mode of derivation and being parallel-free.

Theorem 1. $ME0L^{w,pf} = MET0L^{w,pf} = U \cup \{\emptyset\}$.

Proof. The assertion is obvious, because if an $ME0L$ system is parallel-free, then irrespectively of whatever set/sets of multi-valued functions one chooses, the system produces the same output. So, one can choose the multi-valued functions in such a manner that the productions cannot be applied at all. Therefore, such systems generate languages consisting of at most the axioms of the systems. □

Using a similar argument one can prove the following statement.

Theorem 2. $MT0L^{w,pf} = M0L^{w,pf} = U$.

Next we prove that extended, interactionless M-rate Lindenmayer systems are able to generate any language over a given alphabet V.

Theorem 3. $ME0L^{w} = MET0L^{w} = \mathscr{P}(V^*)$.

Proof. We prove that using these systems we can generate any language L (computable or not). For the sake of simplicity, but without loosing the generality, we might consider only languages over a one letter alphabet. For a given $L \subseteq \{a\}^+$, let us consider the system $H = (V, P, \omega, \Delta, f)$ where $V = \{a, A\}$, $\omega = A$, $\Delta = \{a\}$, and the set P contains the following productions:

$$r_1 : A \to aA,$$
$$r_2 : A \to \lambda,$$
$$r_3 : a \to a.$$

Let us define f as follows:

For a given string $w = a^n A$, $n \geq 0$, we have:

- $f(w) \subseteq \{r_1 r_3^i \mid 0 \leq i \leq n\}$ iff $a^n \notin L$,
- $f(w) \subseteq PS_w^{SAP}$ and there exists $z \in \{r_2 r_3^i \mid 0 \leq i \leq n\}$ such that $z \in f(w)$ iff $a^n \in L$.

It is easy to observe that whenever $a^n \notin L$, then the number of symbol a increases (the production $r_1 : A \to aA$ is executed since there exists $z \in \{r_1 r_3^i \mid 0 \leq i \leq n\}$, such that $z \in f(a^n A)$).

In case of $a^n \in L$, due to the definition of f production $r_2 : A \to \lambda$ might be executed. Therefore, the system H generates a string $w \in L$.

In this way, the constructed system generate any subset of V^*. Hence, we have that $ME0L^{w} = MET0L^{w} = \mathscr{P}(V^*)$. □

In the following we give examples of M-rate $0L$ systems working in the strong mode.

Example 3. Let $H = (V, P, \omega, F)$ be an $M0L$ system working in the *strong* mode such that:

$$V = \{a, b, c\},$$
$$P = \{r_1 : a \to aa,\ r_2 : b \to bb,\ r_3 : c \to cc\},$$
$$\omega = abc,$$
$$f(w) = \{r_1 r_2 r_3\}, \text{ for } w \in V^*.$$

Obviously, H generates the language $L(H) = \{a^n b^n c^n \mid n \geq 1\}$.

Example 4. The language $L = \{a, a^3\}$ is not an $M0L^{s,pf}$ (or $MT0L^{s,pf}$) language. This is proved by contradiction as follows. If there exists an $M0L^{s,pf}$ system $H = (V, P, \omega)$ such that $L(H) = \{a, a^3\}$, then, since obviously $V = \{a\}$, we have two cases: (i) $\omega = a$ and $a \Rightarrow a^3$, hence $a^3 \Rightarrow a^k$, $k \neq 1, 3$, therefore we obtain a contradiction; (ii) $w = a^3$, hence, $a \Rightarrow a$ and $a \Rightarrow \lambda$. Thus $a^3 \Rightarrow a^2$, and so $a^2 \in L(G)$, therefore again we obtain a contradiction.

Obviously, the following results hold.

Proposition 1. $MT0L^{s,pf} \subset RE$.

Proposition 2. $M0L^{s,pf} \subset RE$.

In the following we shall prove that the class of parallel-free MET0L systems in the strong mode identifies the class of ET0L languages.

Theorem 4. $MET0L^{s,pf} = ET0L$.

Proof. We first note that for each $L \in ET0L$ there exists an ET0L system H with two tables such that $L = L(H)$, thus without the loss of generality we can consider ET0L systems with only two tables.
We prove the result by double inclusion.

(1) $MET0L^{s,pf} \supseteq ET0L$
Let us consider an ET0L system $\widetilde{H} = (\widetilde{V}, \widetilde{T}, \widetilde{\omega}, \widetilde{\Delta})$ such that $\widetilde{T} = \{\widetilde{T_1}, \widetilde{T_2}\}$.

Let $h_1 : \widetilde{V}^* \to \overline{V}^*$ be a morphism such that $h_1(a) = \overline{a}$, $a \in \widetilde{V}$. Moreover, let $h_2 : \widetilde{V}^* \to \overline{\overline{V}}^*$ be a morphism such that $h_2(a) = \overline{\overline{a}}$, $a \in \widetilde{V}$.

We will simulate the computation of the system $\widetilde{H}$ with a parallel-free MET0L system $H = (V, T, \omega, \Delta)$ in the strong mode which is defined as follows. Let
- $V = \widetilde{V} \cup \{h_1(A), h_2(A) \mid A \in \widetilde{V}\} \cup \{t_1, t_2, t_3, t_4\} \cup \{\#\}$;
- $T = \{(T_1, f_1), (T_2, f_2), (T_3, f_3), (T_4, f_4)\}$, where

$$
\begin{aligned}
T_1 = {} & \{A \to h_1(\alpha)t_1 \mid A \to \alpha \in \widetilde{T_1}\} \\
& \cup \{h_1(A) \to h_1(A) \mid A \in \widetilde{V}\} \\
& \cup \{h_2(A) \to \# \mid A \in \widetilde{V}\} \\
& \cup \{\# \to \#\,,\ t_1 \to \lambda\,,\ t_2 \to \#\,,\ t_3 \to \#\,,\ t_4 \to \#\}, \\
T_2 = {} & \{A \to A \mid A \in \widetilde{V}\} \\
& \cup \{h_1(A) \to At_2 \mid A \in \widetilde{V}\} \\
& \cup \{h_2(A) \to \# \mid A \in \widetilde{V}\} \\
& \cup \{\# \to \#\,,\ t_1 \to \#\,,\ t_2 \to \lambda\,,\ t_3 \to \#\,,\ t_4 \to \#\}, \\
T_3 = {} & \{A \to h_2(\alpha)t_3 \mid A \to \alpha \in \widetilde{T_2}\} \\
& \cup \{h_1(A) \to \# \mid A \in \widetilde{V}\} \\
& \cup \{h_2(A) \to h_2(A) \mid A \in \widetilde{V}\} \\
& \cup \{\# \to \#\,,\ t_1 \to \#\,,\ t_2 \to \#\,,\ t_3 \to \lambda\,,\ t_4 \to \#\}, \\
T_4 = {} & \{A \to A\,,\ \mid A \in \widetilde{V}\} \\
& \cup \{h_2(A) \to At_4 \mid A \in \widetilde{V}\} \\
& \cup \{h_1(A) \to \# \mid A \in \widetilde{V}\} \\
& \cup \{\# \to \#\,,\ t_1 \to \#\,,\ t_2 \to \#\,,\ t_3 \to \#\,,\ t_4 \to \lambda\};
\end{aligned}
$$

and $f_i : V^* \to \mathscr{P}(R^*)$ with $f_i(x) \in \{X \subseteq RS_x^{MSAP} \mid Pr(X) = Pr(RS_x^{MSAP})\}$ for $x \in V^*$, $1 \leq i \leq 4$, arbitrarily.

- $\omega = \widetilde{\omega}$;
- $\Delta = \widetilde{\Delta}$.

We now show how the construction works. We want to simulate the applications of tables of $\widetilde{H}$; to this aim, without loosing the generality, we may assume that $\widetilde{T_1}$ is simulated first. So, at the beginning, one table is chosen nondeterministically and is applied to the initial sentential form ω. If table T_2 or T_4 is chosen, then the current sentential form is left unchanged (only the productions of type $A \to A$, $A \in V$ are applied). If table T_1 (or T_3) is chosen, then productions of type $A \to h_1(\alpha)t_1$ will be executed. However, because the parallelism is not necessarily total but it depends on the function f_1 (or f_3, respectively), then not necessarily all symbols A from the current sentential form will be rewritten. However, if in the sentential form there is at least one occurrence of symbol A, since in table T_1 there exists a production having the symbol A on the left-hand side and the system works in the strong derivation mode, then A will be rewritten at least once (remember that $f_i(x) \in \{X \subseteq RS_x^{MSAP} \mid Pr(X) = Pr(RS_x^{MSAP})\}$). Consequently, a symbol t_1 is produced. Assume now that still there are symbols A not rewritten by T_1 despite the existence of a production for rewriting A. If this is the case, then if any other table (except T_1) is chosen for a next application, symbol # is generated (hence, the system will not be able anymore to generate a string over Δ because the production $\# \to \#$ is present in every table of H and will always be executed). Therefore, the only table that can be applied and that does not produce the symbol # is again T_1. Finally, all symbols $A \in V$ will be rewritten by applications of table T_1. In addition, table T_1 is also responsible for deleting all symbols t_1. After completing these tasks, the current string will have only images by morphism h_1 of symbols from V (let us call them overlined symbols). At that moment, if we choose any other table except T_2 to apply, then again symbol # will be produced. If table T_2 is applied, then in a similar way as before, the system checks whether or not all overlined symbols are rewritten into the "original" ones. Again, during this checking procedure if we choose to apply a table other than T_2, then symbol # is produced. The simulation of the application of table $\widetilde{T_2}$ follows a similar pattern. In this way, the computation takes place step by step, simulating the computation by $\widetilde{(H)}$ if the proper tables are chosen for application, and always producing the trap symbol if a "wrong" table is chosen for application.

Observe that the strong working mode feature is essential because if at a certain moment a wrong table is chosen for application, then we have to be sure that at least one symbol # is produced, hence a production has to be applied at least once if it can be applied.

In conclusion, we have shown that the constructed system H generates the same language as ET0L system $\widetilde{H}$. Consequently, we have that $MET0L^{s,pf} \supseteq ET0L$.

(2) $ET0L \supseteq MET0L^{s,pf}$

In order to prove this inclusion, we will simulate the computation of an arbitrary MET0L^{s,pf} system $\overline{H} = (\overline{V}, \overline{T}, \overline{\omega}, \overline{\Delta})$ with an appropriate ET0L system $H = (V, T, \omega, \Delta)$. First, we remark that because the system $\overline{H}$ is parallel-free in the

strong mode, then, irrespectively of how the multi-valued functions associated with the sets of productions are chosen, the result of the computation is the same. In particular, one can choose the multi-valued functions such that during the computation the productions are applied in a totally parallel manner (as for the L systems).

Therefore, H is defined as follows:
- $V = \overline{V}$;
- $T = \{T_1, T_2, \ldots, T_k\}$ providing that $\overline{T} = \{\overline{T_1}, \overline{T_2}, \ldots, \overline{T_k}\}$, $k \geq 1$;
- $\omega = \overline{\omega}$;
- $T_i = \{A \to \alpha \mid A \to \alpha \in \overline{T_i}\}$, $1 \leq i \leq k$;
- $\Delta = \overline{\Delta}$.

Observe that in an ET0L system, when a certain table is applied, if a production can be applied then it will be applied (of course, respecting the nondeterminism if it exists). This corresponds to the strong mode of derivation for MET0L systems.

Consequently, we have that $ET0L \supseteq MET0L^{s,pf}$ and therefore we can conclude that $MET0L^{s,pf} = ET0L$. □

4 P Systems with Promoters/Inhibitors

In this section we turn to P systems with promoters/inhibitors.

Definition 5. *A P system with promoters/inhibitors with symbol objects, of degree $m \geq 1$, is a construct*

$$\Pi = (V, \Sigma, C, P, I, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0),$$

where:

- *V is the alphabet of Π; its elements are called objects;*
- *Σ is the output alphabet, $\Sigma \subseteq V$;*
- *$C \subseteq V$ is the set of catalysts;*
- *P is the set of promoters, $P \subseteq V$;*
- *I is the set of inhibitors, $I \subseteq V$;*
- *μ is a membrane structure consisting of m membranes labeled by elements of $1, 2, \ldots, m$;*
- *w_i, $1 \leq i \leq m$, are strings that represent multisets over V associated with the regions $1, 2, \ldots, m$ of μ, called the initial multisets;*
- *R_i, $1 \leq i \leq m$, is a finite set of* evolution rules *associated with the region i of μ.*
  An evolution rule can be non-cooperative, i.e., of the form $a \to v$, or can be promoted (inhibited) non-cooperative, i.e., of the form $a \to v|_p$ (or $a \to v|_{\neg i}$, respectively), it can be catalytic, i.e., of the form $ca \to cv$, or, finally, it can be promoted (inhibited) catalytic, i.e., of the form $ca \to cv|_p$ (or $ca \to cv|_{\neg i}$, respectively), where $a \in (V \setminus C)$, $c \in C$, $p \in P$, $i \in I$, and v is a string representing a multiset over V_{tar}, with $V_{tar} = (V \setminus C) \times TAR$, for $TAR = \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$.

The target is indicated as the index of the object. If no target is specified then is understood as target here*, also, the subscript j is omitted in in_j if the object has only one choice to follow the target indication.*

– $i_0 \in \{1, \ldots, m\}$ *is the label of the output region of* Π.

The *membrane structure* is a hierarchical arrangement of membranes, embedded in a *skin membrane*, the one which separates the system from the environment. Each membrane defines a *region*, i.e., the space in-between the membrane and the membranes directly included in it (if any). Membranes are labeled in order to identify the regions they delimit.

The *configuration* of Π is an instantaneous description of it, including its membrane structure and the contents of all the membranes. The *initial configuration* is composed by the membrane structure μ and the objects initially present in the regions of the system, as described by $w_1, \ldots, w_m$, above.

The result of applying a rule $u \to v$ (or $u \to v|_p$ or $u \to v|_{\neg i}$), where $u, v \in V^*$ is determined by v. If the object a appears in v in the form a_{here}, then it will remain in the same region. If it occurs in v in the form a_{out} or a_{in}, then it will exit to the upper region or to one of the inner regions, respectively.

A promoted rule $u \to v|_p$ can be applied only in the presence of promoter p. An inhibited rule $u \to v|_{\neg i}$ can be applied only in the absence of inhibitor i. Promoters and inhibitors can evolve according to some rules.

For two configurations $C_1 = (w'_1, w'_2, \ldots, w'_m)$ and $C_2 = (w''_1, w''_2, \ldots, w''_m)$ of Π, we say that a *transition* from C_1 to C_2 is performed if we can pass from C_1 to C_2 by applying the rules from $R_1, \ldots, R_m$, in the corresponding regions $1, \ldots, m$, in a maximally parallel manner and with competition on the objects*.

A *computation* of Π is a sequence of transitions between configurations, starting from the initial one. Π performs a *successful computation* iff it halts, i.e., there is no rule applicable to the objects present in the halting configuration.

The *result* of a successful computation is the number (or the vector of numbers) of objects from Σ present in the output region i_0 in the halting configuration. Collecting all the numbers (or vectors of numbers), for any possible halting configuration, we obtain the set $N(\Pi)$ (or $Ps(\Pi)$, respectively) – the set of numbers (vectors of numbers) generated by Π.

The family of sets of numbers (or vectors of numbers) generated by P systems with promoted ($\beta = pro$)/inhibited ($\beta = inh$) non-cooperative ($\alpha = ncoo$) and catalytic ($\alpha = cat_k$) object rewriting rules using at most k catalysts, and having at most m membranes, is denoted by $NOP_m(\alpha, \beta)$ (or $PsOP_m(\alpha, \beta)$, respectively).

The following results regarding the computational power of P systems with promoters/inhibitors are known.

- $PsOP_2(cat_1, pro) = PsOP_2(cat_1, inh) = PsRE$ (see [1],[3]).
- $PsOP_1(ncoo, inh) = PsET0L$ (see [8]).

[1] In Section 5 we consider another mode of applying the rules by employing some computable multi-valued functions that will govern the applications of the rules.

In what follows, we will prove that the model of P systems with non-cooperative promoted rules in computational power equals the model of P systems with non-cooperative inhibited rules.

Our first step is to show that the family of sets of vectors generated by P systems with promoted non-cooperative rules and an arbitrary membrane structure equals the family of sets of vectors generated by P systems with the same features but with only one membrane.

Lemma 1. *$PsOP_m(ncoo, pro) = PsOP_1(ncoo, pro)$, for $m \geq 2$.*

Proof. The inclusion $PsOP_m(ncoo, pro) \supseteq PsOP_1(ncoo, pro)$ is trivial. For the proof of the inclusion $PsOP_m(ncoo, pro) \subseteq PsOP_1(ncoo, pro)$, we construct a P system $\Pi_1 = (V, \Sigma, C, P, I, \mu, w, R, i_0)$ that simulates the computation of the P system $\overline{\Pi_m} = (\overline{V}, \overline{\Sigma}, \overline{C}, \overline{P}, \overline{I}, \overline{\mu}, \overline{w_1}, \ldots, \overline{w_m}, \overline{R_1}, \ldots, \overline{R_m}, \overline{i_0})$ in the following way.

First, let $\mathcal{L} = \{1, 2, \ldots, m\}$ denote the set of labels of the regions of $\overline{\Pi_m}$. Then, we define:

- $V = \{a_i \mid a \in \overline{V}, i \in \mathcal{L}\}$;
- $\Sigma = \{a_i \mid a \in \overline{\Sigma}, i = \overline{i_0} \in \mathcal{L}\}$;
- $C = \overline{C} = I = \overline{I} = \emptyset$;
- $P \subseteq V$;
- $\mu = [\ ]_1$.

Let $h : \overline{V}^* \times \mathcal{L} \to V^*$ be a mapping such that

1) $h(a, i) = a_i, a \in \overline{V}, i \in \mathcal{L}$;
2) $h(\lambda, i) = \lambda$, for all $i \in \mathcal{L}$;
3) $h(x_1x_2, i) = h(x_1, i)h(x_2, i)$, $x_1, x_2 \in \overline{V}^*$, $i \in \mathcal{L}$.

- We define $w = h(\overline{w_1})h(\overline{w_2}) \ldots h(\overline{w_m})$, where $\overline{w_i}$ is the multiset present in region $i \in \mathcal{L}$ of $\overline{\Pi_m}$ at the beginning of the computation;
- R is defined as follows. For each rule $a \to \alpha|_b \in \overline{R_i}$ where $a, b \in \overline{V}$ and α is a string over $\{c, c_{out}, c_{in} \mid c \in \overline{V}\}$, $i \in \mathcal{L}$, to R we add the rule $h(a, i) \to \alpha'|_{h(b,i)}$ where α' is the corresponding string over $\{h(c, i), h(c, j), h(c, k) \mid c \in \overline{V}, i, j, k \in \mathcal{L}\}$, j being the label of the outer region of i, and k being the label of the inner region of i;
- $i_0 = 1$.

In other words, for the P system with a single region that simulates a P system with m regions, we have encoded the region labels into objects (the subscript associated to an object indicates the region the corresponding object belongs to) and we have expressed the rules of regions by the corresponding encoded objects. In this way we ensured that, when simulating $\overline{\Pi_m}$ with Π_1, both the parallelism at the level of regions and at the level of the whole system $\overline{\Pi_m}$ is respected. In addition, one can remark that whenever $\overline{\Pi_m}$ halts, Π_1 halts as well. Moreover, when Π_1 halts, in the output region of Π_1 we will have all the objects corresponding to the multisets present in all regions of $\overline{\Pi_m}$. However, in the output multiset w_{Π_1} of Π_1 we can distinguish the output multiset $w_{\overline{\Pi_m}}$ of

$\overline{\Pi_m}$ because we know which are the objects corresponding to the output region of $\overline{\Pi_m}$ (they are the objects that have as index $\overline{i_0}$).

Therefore, we have that $PsOP_m(ncoo, pro) \subseteq PsOP_1(ncoo, pro)$. Consequently, the theorem holds. □

Now, one can prove that the class of sets of vectors of numbers generated by P systems with non-cooperative promoted rules includes at least the class of Parikh images of languages generated by ET0L systems.

Lemma 2. $PsOP_1(ncoo, pro) \supseteq PsET0L$.

Proof. We will simulate the computation performed by an arbitrary ET0L system $H = (V, T, \omega, \Delta)$ with $T = \{T_1, T_2\}$, using a P system Π_1 defined as follows. (Without the loss of the generality we may assume that H has only two tables.) Let

$$\Pi_1 = (V_\pi, \Sigma, C, P, I, \mu, w, R_\pi, i_0), \text{ where:}$$

- $V_\pi = V \cup \{t, t_1, t_2, K, K_1\}$;
- $\Sigma = \Delta$;
- $C = I = \emptyset$;
- $P \subseteq V$;
- $w = \omega t$;
- $\mu = [\ ]_1$;
- $i_0 = 1$.

The set of rules, R_π, consists of the following elements:

$t \to t_1$,
$t \to t_2$,
$A \to \alpha K|_{t_1}$, for all rules $A \to \alpha \in T_1$,
$A \to \alpha K|_{t_2}$, for all rules $A \to \alpha \in T_2$,
$K \to K_1$,
$t_1 \to t|_A$, for all $A \in V \setminus \Delta$,
$t_2 \to t|_A$, for all $A \in V \setminus \Delta$,
$t_1 \to \lambda|_{K_1}$,
$t_1 \to t|_{K_1}$,
$t_2 \to \lambda|_{K_1}$,
$t_2 \to t|_{K_1}$,
$K_1 \to \lambda$.

At the beginning of the simulation, inside the region of the P system we have the input multiset, consisting of string ω (which corresponds to the axiom of

the ET0L system H), and object t (which represents the starting trigger for the simulation of the nondeterministic table selection mechanism). Nondeterministically, object t is transformed into t_1 or t_2. Once object t_1 (or object t_2) is produced, the simulation of the application of the corresponding table of the ET0L system starts. All rules $A \to \alpha K|_{t_1}$ (or $A \to \alpha K|_{t_2}$, respectively) corresponding to rules $A \to \alpha \in T_1$ (or $A \to \alpha \in T_2$, respectively) are applied in the maximally parallel manner. One can notice that if we applied at least once such a rule, we have produced at least one object K. At this moment, we can distinguish two cases: 1) the current configuration is represented by a multiset that contains objects corresponding to nonterminals of the ET0L system; 2) the current configuration is represented by a multiset that contains only objects corresponding to terminals of the ET0L system.

In the first case one of the rules $t_1 \to t|_A$ or $t_2 \to t|_A$ will be executed, as well as the rule $K \to K_1$. Since an object t is produced, the simulation of applying a table of the ET0L system is iterated (recall that we do not have a terminal string, therefore we do not have to stop).

In the second case, the rules $t_1 \to t|_A$ or $t_2 \to t|_A$ cannot be executed because we assumed that the current configuration is represented by a multiset that contains only objects corresponding to terminals of the ET0L system. Therefore, the rule $K \to K_1$ is executed and afterward one of the rules $t_1 \to \lambda|_{K_1}$ (or $t_2 \to \lambda|_{K_1}$, respectively) and $t_1 \to t|_{K_1}$. Depending on which rule is chosen to be applied we again have two cases – we stop the simulation having a terminal string in the output region or we continue. In both cases, as a last step of the iteration, rule $K_1 \to \lambda$ is applied.

The construction elaborated above proves that $PsOP_1(ncoo, pro) \supseteq PsET0L$. □

For the converse inclusion we have to prove that any P system with non-cooperative promoted rules can be simulated by a P system with non-cooperative inhibited rules. Therefore, since $PsOP_1(ncoo, inh) = PsET0L$, the following result holds.

Lemma 3. $PsOP_1(ncoo, pro) \subseteq PsET0L$.

Proof. Let us consider a P system with non-cooperative promoted rules $\widetilde{\Pi} = (\widetilde{V}, \widetilde{\Sigma}, \widetilde{C}, \widetilde{P}, \widetilde{I}, \widetilde{\mu}, \widetilde{w}, \widetilde{R}, \widetilde{i_0})$ where $\widetilde{V} = \{A_i \mid 1 \leq i \leq k\}$, $\widetilde{\Sigma} \subseteq \widetilde{V}$, $\widetilde{P} \subseteq \widetilde{V}$, $\widetilde{C} = \widetilde{I} = \emptyset$, and $\widetilde{\mu} = [\ ]_1$.

Without any loss of generality we may assume that a non-cooperative rule $A \to \alpha$ is equivalent from computational point of view with the promoted non-cooperative rule $A \to \alpha|_A$. Hence, let us assume that $\widetilde{R} = \widetilde{R_1} \cup \widetilde{R_1} \cup \ldots \cup \widetilde{R_k}$, with $k = card(\widetilde{V})$, where

$$\begin{aligned}\widetilde{R_i} = \{&A_i \to \alpha_{(1,i)}|_{p_{(1,i)}},\\ &A_i \to \alpha_{(2,i)}|_{p_{(2,i)}},\\ &\ldots\ldots\ldots\ldots\\ &A_i \to \alpha_{(t_i,i)}|_{p_{(t_i,i)}}\}\end{aligned}$$

The set $\widetilde{R_i}$, $1 \leq i \leq card(\widetilde{V})$, contains all the rules from $\widetilde{R}$ having the symbol A_i on their left-hand side. Remark that all the rules are promoted by certain objects from $\overline{V}$; in particular all non-cooperative rules were written as the equivalent promoted ones using the above convention.

We construct a P system with non-cooperative inhibited rules Π that simulates the computation of $\widetilde{\Pi}$ as follows. Let

$$\Pi = (V, \Sigma, C, P, I, \mu, w, R, i_0), \text{ where:}$$

- $V = \widetilde{V} \cup \{r_{j,i} \mid A_i \to \alpha_{(j,i)} \in \overline{R}\} \cup \overline{V} \cup \overline{\overline{V}} \cup \dot{V} \cup \ddot{V} \cup \{X_0, X_1, X, Y_0, Y_1, Y\}$;
- $\Sigma = \widetilde{\Sigma}$;
- $C = P = \emptyset$;
- $I \subseteq V$;
- $\mu = [\ ]_1$;
- $i_0 = 1$;

and the set of rules R is defined as follows:

- for each $\widetilde{R_i} \subseteq \widetilde{R}$ defined before we add the following rules [*] to R:

Step	Rule
1	$r_{(j,i)} \to \dot{r}_{(j,i)}\|_{\neg p_{(j,i)}}$, $1 \leq j \leq t_i$
1?	$A_i \to \overline{A_i} X_0$
2?	$X_0 \to X_1 X$
2?	$\overline{A_i} \to \overline{\overline{\alpha_{(j,i)}}} Y_0\|_{\neg \dot{r}_{(j,i)}}$
3?	$\dot{r}_{(j,i)} \to \ddot{r}_{(j,i)}\|_{\neg X_0}$
3?, 4?	$X \to \lambda$
3?	$X_1 \to X$
3?	$Y_0 \to Y_1 Y$
3?	$\overline{\overline{A_i}} \to \dot{A}_i$
3?	$\overline{A_i} \to \dot{A}_i\|_{\neg X_0}$
4?	$\ddot{r}_{(j,i)} \to \lambda\|_{\neg Y}$
4?,5?	$Y \to \lambda$
4?	$Y_1 \to Y$
4?	$\dot{A}_i \to \ddot{A}_i\|_{\neg Y}$
5?	$\ddot{r}_{(j,i)} \to r_{(j,i)}\|_{\neg X}$
5?	$\dot{A}_i \to A_i\|_{\neg X}$

The simulation is performed as follows. First of all, recall that the classical definition of P systems assumes a universal clock that regulates the computation;

[1] On the left-hand side of each rule we have specified the step of a given iteration, in which the corresponding rule might be applied. The question marks indicate that the corresponding rules might not be applicable in that step.

we will use this feature to synchronize different branches of the computation that run in parallel. The basic idea of the simulation is the following: we try to simulate the execution of the rules $A_i \to \alpha_{(j,i)}|_{p_{(j,i)}} \in \overline{R}$ by checking the simultaneous existence of A_i and $p_{(j,i)}$ in different branches of computation (that require some renaming of objects). If they are available simultaneously, then we simulate the rewriting of A_i by $\alpha_{(j,i)}$, otherwise we re-establish the initial configuration (since we have used the renaming of objects). We stop the simulation when we detect that no rules can further be applied in the simulated P system.

In more details, let us consider that with each rule $A_i \to \alpha_{(j,i)}|_{p_{(j,i)}} \in \overline{R}$ an object $r_{(j,i)} \in V$ is associated. Its purpose will be, by means of the rule $r_{(j,i)} \to \dot{r}_{(j,i)}|_{\neg p_{(j,i)}}$, to "check" whether or not the object $p_{(j,i)}$ is in the current multiset (if $p_{(j,i)}$ exists, then in the current multiset we will have the object $\dot{r}_{(j,i)}$). Simultaneously, all objects A_i which are present will be rewritten by the rule $A_i \to \overline{A_i}X_0$. The object X_0 is the root of another branch of the computation that is required to "collect" the symbols $\overline{A_i}$ not rewritten by the rule $\overline{A_i} \to \overline{\overline{\alpha_{(j,i)}}}$ (this situation occurs when the rules of type $A_i \to \alpha_{(j,i)}|_{p_{(j,i)}}$ were not executed because the objects $p_{(j,i)}$ were missing and there were no non-cooperative rules of type $A_i \to \alpha \in \overline{R}$). These objects are rewritten by the rule $\overline{A_i} \to \dot{A}_i$. Later, objects $\dot{A}_i$ will be rewritten into A_i if there exists at least one rule of $\overline{\Pi}$ that was simulated and the process is repeated, otherwise we stop the computation by deleting all objects $\ddot{r}_{(j,i)}$ (by the rule $\ddot{r}_{(j,i)} \to \lambda|_{\neg Y}$) and transforming the objects $\dot{A}_i$ into $\ddot{A}_i$ (by the rule $\dot{A}_i \to \ddot{A}_i|_{\neg Y}$). It is worth mentioning that the objects Y_0 (and their descendants, the objects Y_1, Y) are used as witnesses to the simulation of rules $A_i \to \alpha_{(j,i)}$. If at least one object Y appeared, it means that the simulation should continue since at least one rule was simulated. □

By Lemma 1, Lemma 2, and Lemma 3 we conclude that:

Theorem 5. *$PsOP_m(ncoo, pro) = PsOP_m(ncoo, inh) = PsET0L$, for $m \geq 1$.*

5 On Dynamical Parallelism in P Systems with Non-cooperative Promoted Rules

In this section we will extend the original definition of P systems with symbol rewriting rules to P systems working with dynamical parallelism in rule applications and we will mainly focus on the definition of P systems using non-cooperative promoted rules. However, the notions presented below can easily be extended to other models of P systems.

For instance, considering an alphabet of objects V, a multiset of objects $w \in V^*$, and a set of multiset rewriting rules $R = \{r_i : u_i \to v_i|_{p_i} \mid u_i, v_i, p_i \in V^*, 1 \leq i \leq k\}$ we can define $R_w^{sap} = r_1^{t_1} r_2^{t_2} \ldots r_k^{t_k}$, $t_i \in I\!N$, $1 \leq i \leq k$, a multiset over $R = \{r_i : u_i \to v_i|_{p_i} \mid 1 \leq i \leq k\}$ of multiset rewriting promoted rules simultaneously applicable to w. R_w^{sap} is any multiset such that:

$$\bigcup_{1 \leq i \leq k} t_i * left(r_i) \subseteq w \text{ and } p_i \subseteq w \text{ if } t_i > 0.$$

Based on this concept, as before we can define, the notions R_w^{SAP}, R_w^{MSAP}, D_x, R_w^{MAX}, W_w. Hence, we can consider M-rate derivations in the weak and in the strong mode, in a similar fashion as we did for M0L systems.

Consider a P system $\Pi = (V, \Sigma, C, P, I, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, i_0)$ defined using the standard notation. Let $f_i : V^* \to \mathscr{P}(R^*)$ with $f_i(x) \in \mathscr{P}(R_x^{SAP})$ for $x \in V^*$,, $1 \leq i \leq n$, be computable multi-valued functions.

Now consider the system

$$\Pi(f_1, \ldots, f_n) = (V, C, \Sigma, P, I, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, i_0, f_1, \ldots, f_n)$$

and let us describe how the computations are performed.

Starting from the initial configuration given by the n-tuple $(w_1, \ldots, w_n)$, where w_i, $1 \leq i \leq n$, are multisets over V, the system evolves according to the rules and objects present in the membranes, in a non-deterministic parallel manner. The selection of the rules as well as the number of applications of the selected rules on the multiset w_i, $1 \leq i \leq n$, are given by a multi-valued function f_i, $1 \leq i \leq n$ (we have that $f_i(w_i) \in \mathscr{P}((R_i)_{w_i}^{SAP})$, in case of weak mode derivation, or $f_i(w_i) \in \{X \subseteq R_{w_i}^{MSAP} \mid Pr(X) = Pr(R_{w_i}^{MSAP})\}$ for the strong mode derivation). As usually, the system performs computation steps according to a universal clock.

For a given configuration $(x_1, x_2, \ldots, x_n)$ with x_i, $1 \leq i \leq n$, being multisets over V, the rules are applied according to $(R_i)_{x_i}^{sap} = r_{(i,1)}^{t_1} r_{(i,2)}^{t_2} \cdots r_{(i,k)}^{t_k} \in f_i(x_i)$, $1 \leq i \leq n$ (i.e., given a multiset x_i, the rule $r_{(i,j)}$ is applied t_j times, $1 \leq j \leq k$).

For two configurations $C_1 = w'$ and $C_2 = w''$ of Π, we say that a transition is performed from C_1 to C_2 if we can pass from C_1 to C_2 by using the evolution rules from R_i, $1 \leq i \leq n$, applied according to the functions f_i, $1 \leq i \leq n$.

As usual, a *computation* of a P system Π is a sequence of transitions between configurations. The system will make a *successful computation* if and only if it halts. In case of generative P systems, the *result* of a successful computation is the number (or the vector of numbers) of objects from Σ present in the membrane i_0, in a halting configuration of Π. If the computation never halts, then we will have no output.

A P system Π is *parallel-free* if and only if for any set of functions f_i, $1 \leq i \leq n$, the system Π produces the same set of vectors of natural numbers.

For such P systems, when speaking about the generated families of sets of (vectors of) natural numbers we will specify the mode of derivation (weak or strong), as well as the parallel-free property by superscripts associated to the classical notation given in Section 4. For instance, $PsOP_m^{s,pf}(ncoo, pro)$ represents the family of sets of numbers generated by P systems using non-cooperative promoted rules, working in the strong mode and having the parallel-free property.

Let us consider the following example:

Example 5. Let the P system $\Pi = (V, \Sigma, C, P, I, \mu, w, R, \vartheta, f)$ be defined as follows:

$$V = \{a, p\};$$
$$\Sigma = \{a\};$$

$$
\begin{aligned}
&C = \emptyset;\\
&P = \{p\};\\
&I = \emptyset;\\
&\mu = [\ ]_1;\\
&w = a^2p;\\
&R = \{r_1 : a \to aaa|_p, r_2 : p \to p, r_3 : p \to \lambda\};\\
&\vartheta = 1;
\end{aligned}
$$

$f(x) = \{r_1^{[0.5*|x|_a]+1} r_2^i r_3^j \mid i,j \in \{0,1\}, i+j=1\}$, $x \in V^*$.

Let us investigate in more details how this system performs computation. First, observe that the execution of rule r_1 is controlled by the promoter p, while the number of applications of r_1 on the current multiset is given by the function f. Assume that the rule $r_1 : a \to aaa|_p$ is executed k times in k derivation steps. Then we have:

$$
\begin{aligned}
|w|_a = |w_1|_a &= 2;\\
|w_2|_a &= 6;\\
|w_3|_a &= 14;\\
&\cdots\cdots\cdots\cdots\cdots\cdots\\
|w_k|_a &= ([|w_{k-1}|_a * 0.5] + 1) * card(right(r_1))\\
&+ |w_{k-1}|_a - ([|w_{k-1}| * 0.5] + 1)\\
&= 2 * (1 + [|w_{k-1}| * 0.5]) + |w_{k-1}|_a
\end{aligned}
$$

where w_i denotes the multiset of objects present in the region of Π, at the step i of computation.

Observe that if $|w_1|_a \vdots 2 \Rightarrow |w_k|_a \vdots 2$, then this means that $[|w_i|_a * 0.5] = |w_i|_a * 0.5$, $1 \leq i \leq k$. Since $|w_1|_a = 2$, we can drop off the integer part function and we have the following recurrent formulas:

$$
\begin{array}{rcl|l}
|w_k|_a &=& 2 * |w_{k-1}|_a + 2 & *2^0\\
|w_{k-1}|_a &=& 2 * |w_{k-2}|_a + 2 & *2^1\\
\cdots\cdots & & \cdots\cdots\cdots\cdots & \\
|w_2|_a &=& 2 * |w_1|_a + 2 & *2^{k-2}
\end{array}
$$

In order to obtain the general term $|w_k|_a$, we will multiply each recurrent formula by a corresponding constant (as shown above) and we will sum up the results. It follows that

$|w_k|_a = 2^{k-1} * |w_1|_a + 2^1 + 2^2 + \cdots + 2^{k-1} = 2 + \cdots + 2^k = \frac{2*(2^k-1)}{2-1} = 2^{k+1} - 2$.

This means that Π generates the set $\{2^{k+1} - 2 \mid k \geq 2\}$.

In Section 3 we have shown that $MET0L^{s,pf} = ET0L$. Since we have proved in a constructive manner that $PsOP_m(ncoo, pro) = PsOP_m(ncoo, inh)$, for $m \geq 1$, and in [8] it is proved also constructively that $PsOP_m(ncoo, inh) = PsET0L$, then the following result holds.

P systems with $m \geq 1$ membranes, using non-cooperative promoted symbol objects rewriting rules, working in the strong mode, and having the parallel-free property, generate the set of all Parikh images of languages generated by ET0L systems. More formally, we state:

Theorem 6. *$PsOP_m^{s,pf}(ncoo, pro) = PsET0L$, for $m \geq 1$.*

6 Conclusions

In this paper we have considered a general perspective of modeling biological systems. The chemical reactions that might take place in real organisms are modeled by rewriting objects but not in a totally parallel manner as in the case of Lindenmayer systems or maximally parallel manner as in the P systems framework.

We define the parallelism with the help of multi-valued functions that for a given configuration establish the number of times the rules are applied. Regarding this matter, we propose two ways of performing the derivation, the *weak* and the *strong* modes, respectively. The *weak mode* describes the case when no restriction is imposed on the way the rewriting is done; the system is parallel (not necessarily maximal) with competition on objects if it is the case. The *strong mode* of derivation assumes that rules are applied in a manner described by the multi-valued functions, but, in addition one can remark that each distinct symbol from the sentential form is rewritten at least once (of course, if there are rules that can handle it).

We have also shown that the new classes of P systems contain proper subclasses which are able to generate/accept the same families of sets of vectors/ numbers as the corresponding upper classes, regardless the choice of the rewriting parallelism. The systems having this property are called *parallel-free* P systems.

Several results in these topics have been presented. In addition, we have studied known problems in P systems theory. In Section 4 we have solved an open problem in [4], regarding the computational power of P systems with non-cooperative promoted rules.

Acknowledgments

The work of the third author was possible due to a doctoral FPU grant from Ministry of Education, Spain.

References

1. P. Bottoni, C. Martín-Vide, Gh. Păun, G. Rozenberg, Membrane Systems with Promoters/Inhibitors. *Acta Informatica*, 38, 10 (2002), 695–720.
2. M. Cavaliere, D. Sburlan, Time Independent P Systems, *Lecture Notes in Computer Science*, 3365 (2005), 239–258.
3. M. Ionescu, D. Sburlan, On P Systems with Promoters/Inhibitors, *International Journal of Universal Computer Science*, 10, 5 (2004), 581–599.

4. Gh. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
5. G. Rozenberg, A. Salomaa, *The Mathematical Theory of L Systems*, Academic Press, New York, 1980.
6. G. Rozenberg, A. Salomaa, Eds., *Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.
7. D. Sburlan, *Promoting and Inhibiting Contexts in Membrane Systems*, doctoral Thesis, University of Seville, Spain, 2006.
8. D. Sburlan, Further Results on P Systems with Promoters/Inhibitors, *International Journal of Foundations of Computer Science*, 17, 1 (2006), 205–222.

P Colonies with a Bounded Number of Cells and Programs*

Erzsébet Csuhaj-Varjú[1,2], Maurice Margenstern[3], and György Vaszil[1]

[1] Computer and Automation Research Institute, Hungarian Academy of Sciences
Kende utca 13–17, H-1111 Budapest, Hungary
{csuhaj,vaszil}@sztaki.hu
[2] Department of Algorithms and Their Applications, Loránd Eötvös University
Pázmány Péter sétány 1/c, H-1117 Budapest, Hungary
[3] Université Paul Verlaine - Metz, LITA, EA 3097
Île du Saulcy, 57045 Metz Cedex 1, France
margens@univ-metz.fr

Abstract. We continue the investigation of P colonies, a class of abstract computing devices composed of very simple agents (computational tools), acting and evolving in a shared environment. We show that if P colonies are initialized by placing a number of copies of a certain object in the environment, then they can generate any recursively enumerable set of numbers with a bounded number of cells, each cell containing a bounded number of programs (of bounded length), for constant bounds.

1 Introduction

P colonies were introduced in [4] as a class of very simple membrane systems similar to the so-called colonies of simple formal grammars ([3]). A P colony intends to model a community of very simple cells living together in a shared environment. The cells are represented by a collection of objects and rules for processing these objects, they are the basic computing agents in this formal model of computing.

To restrict the capabilities of the agents, only two (or three) objects are allowed to be inside any cell. Moreover, the rules of the cells are either of the form $a \to b$, specifying that an internal object a is transformed into an internal object b, or of the form $c \leftrightarrow d$, specifying the fact that an internal object c is sent out of the cell, to the environment, in exchange of the object d, which was present in the environment and is now brought inside the cell. Thus, a cell containing the objects a, c will contain the objects b, d after applying these rules in parallel. Two such rules can also be combined into so called "checking" rules of the form

* This publication was supported by the Hungarian Foundation for Research and Technological Innovation (project no. TéT F-19/04) and the EGIDE in France (project no. Balaton 09000TC, year 2005) in the frame of the Hungarian-French Intergovernmental Scientific and Technological Cooperation. The work of Erzsébet Csuhaj-Varjú and György Vaszil was also supported by the Hungarian Scientific Research Fund "OTKA" grant no. T 042529.

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 352–366, 2006.

$c \leftrightarrow d/c' \leftrightarrow d'$ or $c \leftrightarrow d/a \rightarrow b$ which specify two possible actions: if the first rule is not applicable then the second one should be applied.

With each cell, we associate a set of programs composed of rules as above. In the case of systems consisting of cells with only two objects inside, each program has two rules; when considering cells with three objects inside, then the programs have three rules. The rules of the program must be applied in parallel to the objects in the cell.

The cells of a P colony execute a computation by synchronously applying their programs to objects inside the cells and outside in the environment. When a halting configuration is reached, that is, when no more rules can be applied, the result of the computation is read as the number of certain types of objects present in the environment.

For more information on membrane computing, consult the monograph [7], for more on grammar systems and colonies in particular, see [1] and [3], respectively.

It was shown in [4] and [2] that P colonies are able to compute any recursively enumerable set of numbers, even in the situation when the starting configuration contains an infinite number of copies of a certain object in the environment, and two or three copies of the same object inside the cells. However, there is no lower bound given for the number of cells or the number of programs needed to reach this power.

In the present paper we show that if the environment is initialized by a finite multiset of objects before the computation begins, then P colonies generate any recursively enumerable set of natural numbers with a bounded number of cells and a bounded number of programs in each cell, for constant bounds. The values of the bounds presented in our results depend on the type of rules the P colonies are given with and they suggest a trade-off between the number of necessary cells and the number of necessary programs in each cell. Our results demonstrate that one cell with a bounded, but fairly large amount of programs, might possess the power of Turing machines, which power can also be reached by several cells and a significantly lower number of programs in each cell.

2 Preliminaries and Definitions

Let V be an alphabet, let V^* be the set of all words over V, and let ε denote the empty word. We denote the length of a word $w \in V^*$ by $|w|$, and the number of occurrences of a symbol $a \in V$ in w by $|w|_a$. The set of non-negative integers is denoted by $\mathbb{N}$.

A multiset over an arbitrary (not necessarily finite) set V is a mapping $M : V \rightarrow \mathbb{N}$ which assigns to each object $a \in V$ its multiplicity $M(a)$ in M. The support of M is the set $supp(M) = \{a \mid M(a) \geq 1\}$. If V is a finite set, then M is called a finite multiset. The set of all finite multisets over the finite set V is denoted by V°. We say that $a \in M$ if $M(a) \geq 1$, the cardinality of M, $card(M)$ is defined as $card(M) = \Sigma_{a \in M} M(a)$. For two multisets $M_1, M_2 : V \rightarrow \mathbb{N}$, $M_1 \subseteq M_2$ holds, if for all $a \in V$, $M_1(a) \leq M_2(a)$. The union of M_1 and M_2 is defined as $(M_1 \cup M_2) : V \rightarrow \mathbb{N}$ with $(M_1 \cup M_2)(a) = M_1(a) + M_2(a)$ for all

$a \in V$, the difference is defined for $M_2 \subseteq M_1$ as $(M_1 - M_2) : V \to \mathbb{N}$ with $(M_1 - M_2)(a) = M_1(a) - M_2(a)$ for all $a \in V$. A multiset M is empty if its support is empty, $supp(M) = \emptyset$.

We will represent a finite multiset M over V by a string w over the alphabet V with $|w|_a = M(a)$, $a \in V$, and ϵ will represent the empty multiset.

Now we recall the definition of a P colony [4]. A *P colony* is a construct

$$\Pi = (V, e, o_f, I_E, C_1, \ldots, C_n),\ n \geq 1,$$

where V is an alphabet (its elements are called *objects*), e (the environmental object) and o_f (the final object) are two distinguished objects of V, $I_E \in (V - \{e\})^\circ$ is a finite multiset of objects initially present in the environment besides the infinitely many copies of e, and $C_1, \ldots, C_n$ are the *cells* of the colony. In [2] and in [4], the multiset I_E is empty, that is, initially only an infinite supply of e objects are present in the environment. Here we define I_E to be a finite non-empty multiset, because we would like to allow the initialization of the colony by placing objects different from e in the environment.

Each cell C_i, $1 \leq i \leq n$, is a pair $C_i = (O_i, P_i)$, where O_i is a multiset over $\{e\}$ having the same cardinality for all i, $1 \leq i \leq n$ (the initial state of the cell), and P_i is a finite set of *programs*; each program being a set of rules of the forms $a \to b$ (internal point mutation), $c \leftrightarrow d$ (one object exchange with the environment), $c \leftrightarrow d/c' \leftrightarrow d'$ (checking rule for one object exchange with the environment), or $c \leftrightarrow d/a \to b$ (checking rule for one object exchange with the environment or internal point mutation), where $a, b, c, d, c', d' \in V$. The programs contain one rule for each element of O_i, thus, the number of rules of a program coincides with the cardinality of O_i, $1 \leq i \leq n$.

A program is called restricted if it contains one point mutation rule of the form $a \to b$, and either one exchange rule of the form $c \leftrightarrow d$, or one checking rule of the form $c \leftrightarrow d/c' \leftrightarrow d'$. A P colony is called restricted if it contains two objects in each cell and has only restricted programs. Two-objects colonies with non-restricted programs, or three-objects colonies are called non-restricted.

The programs of the cells are used in the non-deterministic maximally parallel way usual in membrane computing: in each time unit, each cell which can use one of its programs should use one. When using a program, each of its rules must be applied to distinct objects of the cell. In this way, we get transitions among the configurations of the colony. A sequence of transitions is a *computation*. A computation is halting if it reaches a configuration where no cell can use any program. The result of a halting computation is the number of copies of the object o_f present in the environment in the halting configuration. Initially, the environment contains I_E, a finite number of copies of objects from $V - \{e\}$ together with arbitrarily many copies of the environmental object e. Moreover, as stated above, the cells also contain two or three copies of e inside.

Because of the non-determinism in choosing the programs, several computations can be obtained from a given initial configuration, hence with a P colony Π we can associate a set of numbers computed by all possible halting computations of Π.

For a P colony $\Pi = (V, e, o_f, I_E, C_1, \ldots, C_n)$ as above, a configuration can be formally written as an $(n+1)$-tuple

$$(w_1, \ldots, w_n; w_E),$$

where w_i represents the multiset of objects from cell C_i, $1 \leq i \leq n$ ($w_i \in V^2$ in the case of two-objects colonies and $w_i \in V^3$ in the case of three-objects colonies), and $w_E \in (V - \{e\})^*$ represents the multiset of objects from the environment different from the "background" object e.

The initial configuration is $(ee, \ldots, ee; I_E)$ in the case of two-objects colonies and $(eee, \ldots, eee; I_E)$ in the case of three-objects colonies where $I_E \in (V - \{e\})^*$.

Let the programs of each P_i be labeled in a one-to-one manner by labels in the set $lab(P_i)$ in such a way that $lab(P_i) \cap lab(P_j) = \emptyset$ for $i \neq j$, $1 \leq i, j \leq n$. For a rule r and a multiset $w \in V^\circ$, let $left(r, w) = a$ and $right(r, w) = b$ if $a \in w$ and r is a point mutation rule $r = (a \to b)$ or a checking rule $r = (c \leftrightarrow d/a \to b)$ and $d \notin w$, and let $left(r, w) = right(r, w) = \epsilon$ otherwise. Let also, for a rule r and a multiset $w \in V^\circ$, $export(r, w) = c$ and $import(r, w) = d$ if $d \in w$ and r is an exchange rule $r = (c \leftrightarrow d)$ or a checking rule $r = (c \leftrightarrow d/c' \leftrightarrow d')$. If r is a checking rule as above with $d \notin w$ but $d' \in w$, then let $export(r, w) = c'$, $import(r, w) = d'$. Let $export(r, w) = import(r, w) = \epsilon$ in all other cases. For a program p and any $\alpha \in \{left, right, export, import\}$, let $\alpha(p, w) = \bigcup_{r \in p} \alpha(r)$.

A transition from a configuration to another is denoted as

$$(w_1, \ldots, w_n; w_E) \Rightarrow (w'_1, \ldots, w'_n; w'_E)$$

where the following conditions are satisfied: There is a set of program labels P with $|P| \leq n$, such that $p, p' \in P$, $p \neq p'$, $p \in lab(P_j)$ implies $p' \notin lab(P_j)$, and for each $p \in P$, $p \in lab(P_j)$, $left(p, w_E) \cup export(p, w_E) = w_j$, and $\bigcup_{p \in P} import(p, w_E) \subseteq w_E$. Furthermore, the chosen set P is maximal, that is, if any other program $r \in \bigcup_{1 \leq i \leq n} lab(P_i)$, $r \notin P$, is added to P, then the conditions above are not satisfied.

Now, for each j, $1 \leq j \leq n$, for which there exists a $p \in P$ with $p \in lab(P_j)$, let

$$w'_j = right(p, w_E) \cup import(p, w_E).$$

If there is no $p \in P$ with $p \in lab(P_j)$ for some j, $1 \leq j \leq n$, then let

$$w'_j = w_j,$$

and moreover, let

$$w'_E = w_E - \bigcup_{p \in P} import(p, w_E) \cup \bigcup_{p \in P} export(p, w_E).$$

A configuration is halting if the set of program labels P satisfying the conditions above cannot be chosen to be other than the empty set, $\emptyset$.

The set of numbers computed by a P colony Π is defined as

$$N(\Pi) = \{|v_E|_{o_f} \mid (w_1, \ldots, w_n, I_E) \Rightarrow^* (v_1, \ldots, v_n, v_E)\}$$

where $(w_1, \ldots, w_n, I_E)$ is the initial configuration, $(v_1, \ldots, v_n, v_E)$ is a halting configuration, and $\Rightarrow^*$ denotes the reflexive and transitive closure of $\Rightarrow$.

The family of all sets of numbers computed as above by initialized environment P colonies with k-objects ($k = 2, 3$) of degree at most $n \geq 1$ having at most $h \geq 1$ programs in the cells without using checking rules, is denoted by $IPCol\_k(n, h)$. In the case of two-objects colonies, if we use only restricted programs, then we write $IPCol\_2R$ instead of $IPCol\_2$. If checking rules are allowed, then we write $IPCol\_Ch\_k$ instead of $IPCol\_k$; thus, for instance, $IPCol\_Ch\_2R(n, h)$ will be the family of numbers computed by restricted two-objects P colonies with at most n cells, each having at most h programs where the use of checking rules is allowed.

In the following we compare the families $IPCol\_\alpha(n, h)$, where $\alpha \in \{2, 3, 2R, Ch\_2, Ch\_3, Ch\_2R\}$, with $\mathbb{N}RE$, the family of sets of numbers computed by Turing machines, and for this we need the notion of a (non-deterministic) *register machine*. A register machine consists of a given number of registers each of which can hold an arbitrarily large non-negative integer number (we say that the register is empty if it holds the value zero), and a set of labeled instructions which specify how the numbers stored in registers can be manipulated. There are several types of instructions which can be used.

- $l_i : (\texttt{ADD}(r), l_j, l_k)$ - add 1 to register r and then go to one of the instructions with labels l_j or l_k, non-deterministically chosen,
- $l_i : (\texttt{SUB}(r), l_j)$ - if register r is non-empty, then subtract 1 from it, otherwise leave it unchanged, and go to the instruction with label l_j in both cases,
- $l_i : (\texttt{CHECK}(\texttt{r}), l_j, l_k)$ - if the value of register r is zero, go to instruction l_j, otherwise go to l_k,
- $l_i : (\texttt{CHECKSUB}(r), l_j, l_k)$ - if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k,
- $l_h : \texttt{HALT}$ - halt the machine.

Thus, formally, a *register machine* is a construct $M = (m, H, l_0, l_h, R)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halting label, and R is the set of instructions; each label from H labels only one instruction from R. A register machine M computes a set $N(M)$ of numbers in the following way: it starts with empty registers by executing the instruction with label l_0 and proceeds by applying instructions as indicated by the labels (and made possible by the contents of the registers); if the halt instruction is reached, then the number stored at that time in register 1 is said to be computed by M. Because of the non-determinism in choosing the continuation of the computation in the case of ADD instructions, $N(M)$ can be an infinite set.

It is known (see, e.g., [6]) that in this way we can compute all sets of numbers which are Turing computable by using instructions of type ADD, CHECKSUB, and HALT. It is also known, that there exist universal register machines with a small number of registers and a small number of instructions, the exact numbers depending on the chosen set of instructions and the chosen notion of universality.

In [5] several results on small universal register machines are presented. The register machines in this framework are used to compute functions of non-negative integers by having the argument of the function in one of the registers before the computation starts, and obtaining the result of the function in an other register after a halting computation. The universal machines have eight registers, and they can simulate the computation of any register machine M with the help of a "program", an integer $code(M) \in \mathbb{N}$ coding the particular machine M. If $code(M)$ is placed in the second register and an argument $x \in \mathbb{N}$ is placed in the third register, then the universal machine simulates the computation of M by halting if and only if M halts, and by producing the same result in its first register as M produces in its output register after a halting computation.

Since these machines are defined to compute functions of non-negative integers and work in such a way that the argument of the function is initially present in the third register, we need to modify them in order to conform to the number generating definition we have given above. Thus, we add a new start label l'_0 and a separate non-deterministic ADD instruction, $l'_0 : (\texttt{ADD}(r_3), l'_0, l_0)$, to produce an argument $x \in \mathbb{N}$ in the third register before the actual computation begins, that is, to make the resulting universal machine generate any value from the range of the function computed by the simulated register machine. We summarize the results we use from [5] in the following theorem.

Theorem 1. [5] *Let $\mathbb{M}$ be the set of register machines. Then, there are register machines U_1, U_2, U_3 with eight registers and a recursive function $g : \mathbb{M} \rightarrow \mathbb{N}$ such that for each $M \in \mathbb{M}$, $N(M) = N(U_i(g(M)))$, where $N(U_i(g(M)))$ denotes the set of numbers computed by U_i, $1 \leq i \leq 3$, with initially containing $g(M)$ in the second register.*

All these machines have one HALT *instruction labeled by l_h, one instruction of the type* ADD *labeled by l_0, and*

- *U_1 has $8 + 11 + 13 = 32$ instructions of the types* ADD, SUB, *and* CHECK, *respectively,*
- *U_2 has $9 + 13 = 22$ instructions of the types* ADD, *and* CHECKSUB, *respectively,*
- *U_3 has $8 + 1 + 12 = 21$ instructions of the types* ADD, CHECK, *and* CHECKSUB, *respectively.*

Moreover, these machines either halt using the HALT *instruction and having the result of the computation in the first register, or their computation goes on infinitely.*

3 The Universality of P Colonies with a Bounded Number of Cells

The proofs of the following results are based on techniques from [2] which are combined with Theorem 1.

3.1 Two-Objects P Colonies with Checking Rules

First we consider restricted two-objects P colonies with checking rules.

Theorem 2

$$IPCol\_Ch\_2R(23,5) = IPCol\_Ch\_2R(22,6) = \mathbb{N}RE.$$

Proof. Let $L \in \mathbb{N}RE$ and let M be a register machine with $L = N(M)$. Consider the universal register machines from Theorem 1. By placing the code of M, the value $g(M)$, in the second register, they compute $N(M)$, producing the result in their first register. We show how to construct P colonies which simulate the computation of U_2 and U_3.

Consider a P colony $(V, e, a_1, I_E, C_1, \ldots, C_n)$ where V contains the special object e, two symbols l_i and l_i' for each instruction label l_i of the universal machine, and one symbol a_j, $1 \leq j \leq 8$, for each register. The number of symbols a_j in the environment corresponds to the value of register j. Thus, the initial contents of the environment, I_E consists of $g(M)$ copies of object a_2 and the label of the initial instruction l_0. And so, the result of the computation can be read as the number of objects a_1 corresponding to the value of the first register in a halting configuration.

The P colony we are going to construct contains one cell for each instruction.

An instruction $l_i : (\texttt{ADD}(r), l_{i,1}, l_{i,2})$ ($l_{i,1}, l_{i,2}$ are the labels of the corresponding instructions) is simulated by increasing the number of objects corresponding to the value of register r by one and by changing the instruction label l_i present in the environment to $l_{i,1}$ or $l_{i,2}$. This can be done with five programs as follows.

$$P_i = \{\langle e \to a_r; e \leftrightarrow l_i\rangle,\ \langle l_i \to l_{i,1}; a_r \leftrightarrow e\rangle,\ \langle e \to e; l_{i,1} \leftrightarrow e\rangle,$$
$$\langle l_i \to l_{i,2}; a_r \leftrightarrow e\rangle,\ \langle e \to e; l_{i,2} \leftrightarrow e\rangle\}.$$

The reader can easily verify that the cell C_i starts working when label l_i occurs in the environment. Then, by applying $\langle e \to a_r; e \leftrightarrow l_i\rangle$, the first program listed above in P_i, one copy of symbol a_r is generated (meantime l_i is brought in the cell), and then either by the the second or by the fourth program (i.e., by $\langle l_i \to l_{i,1}; a_r \leftrightarrow e\rangle$ or by $\langle l_i \to l_{i,2}; a_r \leftrightarrow e\rangle$) a_r is exported to the environment. C_i finishes its activity by sending either $l_{i,1}$ or $l_{i,2}$ out from the cell to the environment by using the third or the fifth program given above in P_i, respectively.

An instruction $l_i : (\texttt{CHEKSUB}(r), l_{i,1}, l_{i,2})$ is simulated with five programs by exchanging the label l_i to $l_{i,1}$ and decreasing the number of objects a_r by one, or if the number of objects a_r is zero, then exchanging l_i to $l_{i,2}$.

$$P_i = \{\langle e \to e; e \leftrightarrow l_i\rangle,\ \langle l_i \to l_{i,1}; e \leftrightarrow a_r / e \leftrightarrow e\rangle,\ \langle a_r \to e; l_{i,1} \leftrightarrow e\rangle,$$
$$\langle l_{i,1} \to l_{i,2}; e \leftrightarrow e\rangle,\ \langle e \to e; l_{i,2} \leftrightarrow e\rangle\}.$$

As in the previous case, the cell starts working when label l_i appears in the environment. Then, by using the first program in P_i, above, l_i is brought in the

cell. If the environment contains at least one copy of a_r, then one a_r is brought inside the cell (by using $\langle l_i \to l_{i,1}; e \leftrightarrow a_r / e \leftrightarrow e\rangle$, the second program given in P_i) and meantime l_i is changed for $l_{i,1}$. If the environment is free from a_r, then inside the cell only l_i is changed for $l_{i,1}$. The cell finishes its activity either by applying the third program (i.e., $\langle a_r \to e; l_{i,1} \leftrightarrow e\rangle$) which changes a_r to e and sends label $l_{i,1}$ to the environment or by using the fourth and after then the fifth program when label $l_{i,2}$ is sent out from the cell, indicating that no a_r was present in the environment. The computation continues with the simulation of the instruction labeled with $l_{i,2}$.

An instruction of type $l_i : (\texttt{CHECK}(r), l_{i,1}, l_{i,2})$ is simulated by six programs as follows.

$$P_i = \{\langle e \to e; e \leftrightarrow l_i\rangle,\ \langle l_i \to l'_{i,1}; e \leftrightarrow a_r / e \leftrightarrow e\rangle,\ \langle l'_{i,1} \to l_{i,1}; a_r \leftrightarrow e\rangle,$$
$$\langle e \to e; l_{i,1} \leftrightarrow e\rangle,\ \langle l'_{i,1} \to l_{i,2}; e \leftrightarrow e\rangle,\ \langle e \to e; l_{i,2} \leftrightarrow e\rangle\}.$$

This cell works similarly to the previous one, the only difference is that after bringing one copy of a_r inside the cell, this symbol must be returned to the environment, which is performed by applying the third program (i.e., by $\langle l'_{i,1} \to l_{i,1}; a_r \leftrightarrow e\rangle$).

In the colony, there are no programs for the halting label l_h, thus, its appearance ends the computation which otherwise never stops.

For the simulation of U_2, consider $\Pi_2 = (V, e, a_1, I_E, C_0, \ldots, C_{22})$ with V and I_E as above, and one cell with the programs P_0 for the initial ADD instruction labeled with l_0 which fills the input register, nine cells with $P_i,\ 1 \leq i \leq 9$, for the simulation of the rest of the ADD instructions, and thirteen cells with $P_i,\ 10 \leq i \leq 22$, for simulating the CHECKSUB instructions. This gives us $1 + 9 + 13 = 23$ cells with at most 5 programs.

For the simulation of U_3, consider $\Pi_3 = (V, e, a_1, I_E, C_0, \ldots, C_{21})$ with V, I_E and P_0 as above, eight cells with $P_i,\ 1 \leq i \leq 8$, for the simulation of the ADD instructions, one cell with P_9 for the CHECK instruction, and twelve cells with $P_i,\ 10 \leq i \leq 21$, for simulating the CHECKSUB instructions. This gives us $1 + 8 + 1 + 12 = 22$ cells, this time with at most 6 programs.

Thus, we have shown that U_2 can be simulated with 23 cells having at most 5 programs, and that U_3 can be simulated with 22 cells having at most 6 programs. This proves our statement. □

The number of programs in one cell can be decreased if we use arbitrary, not necessarily restricted programs.

Theorem 3

$$IPCol\_Ch\_2(22, 5) = \mathbb{N}RE.$$

Proof. Consider U_3 from Theorem 1 and $\Pi_3 = (V, e, a_1, I_E, C_0, \ldots, C_{21})$ from the proof of Theorem 2 where cell C_9 corresponds to the instruction $l_9 : (\texttt{CHECK}(r), l_{9,1}, l_{9,2})$ which is simulated in P_9 with six restricted programs as

described above. If we allow non-restricted programs, then we can replace P_9 with

$$P'_9 = \{\langle e \to e; e \leftrightarrow l_9\rangle,\ \langle l_9 \to l_{9,1}; e \leftrightarrow a_r / e \to l_{9,2}\rangle,\ \langle l_{9,1} \leftrightarrow e; a_r \leftrightarrow e\rangle,$$
$$\langle l_{9,1} \to e; l_{9,2} \leftrightarrow e\rangle\}$$

which achieves the same effect as P_9 with four programs. After bringing l_9 inside the cell (by performing the first program given in P'_9), l_9 is rewritten to $l_{9,1}$ and depending on whether or not the environment contains at least one occurrence of a_r, either one copy of a_r is brought inside the cell or e is changed for $l_{9,2}$. In the first case both $l_{9,1}$ and a_r are exported to the environment, thus the presence of a_r in the environment is verified and the computation continues with simulating the instruction labeled by $l_{9,1}$. The second case proves that the environment is free from a_r and then the computation continues with simulating the instruction labeled with $l_{9,2}$, after exporting $l_{9,2}$ to the environment. Thus, we can simulate U_3 with $\Pi'_3 = (V, e, a_1, I_E, C_0, \ldots, C_8, C'_9, C_{10}, \ldots, C_{21})$, $C'_9 = (O_9, P'_9)$, which gives us 22 cells with at most 5 programs. As 5 programs are necessary for the simulation of the `ADD` and the `CHECKSUB` instructions, at most 5 programs are needed for each P_i. □

Now we show that by increasing the number of programs, one cell is sufficient to generate any set in $\mathbb{N}RE$, even with restricted programs.

Theorem 4

$$IPCol\_Ch\_2R(1, 142) = \mathbb{N}RE.$$

Proof. Similarly to the proof of Theorem 2, we show how to construct a P colony Π which simulates the computation of the universal register machine U_3 from Theorem 1.

Let the nine `ADD` instructions be labeled by $l_0, \ldots, l_8$, the one `CHECK` instruction be labeled by l_9, and the twelve `CHECKSUB` instructions be labeled by $l_{10}, \ldots, l_{21}$.

Let $\Pi = (V, e, a_1, I_E, C)$ where V contains the special objects e, t; the symbol l_i for each instruction of U_3, that is, for $0 \le i \le 21$; the symbols l'_i for the `ADD` instructions, that is, for $0 \le i \le 8$; the symbols l''_i for the `CHECK` and `CHECKSUB` instructions, that is, for $9 \le i \le 21$; and one symbol a_j for each register j, $1 \le j \le 8$, of U_3. The initial contents of the environment, I_E consists of a number of copies of the object a_2 for initializing the contents of the second register, and one copy of the symbols l'_i for $0 \le i \le 8$. The result of a computation can be read in a halting configuration as the number of objects a_1 in the environment.

The computation starts by producing an arbitrary number of copies of the double primed instruction labels l''_i, $9 \le i \le 21$, and the introduction of a copy of the initial instruction label l_0. This is achieved with the following programs.

$$P_{ini} = \{\langle e \to l''_9; e \leftrightarrow e\rangle,\ \langle e \to l''_{21}; l''_{21} \leftrightarrow e\rangle,\ \langle e \to l_0; l''_{21} \leftrightarrow e\rangle\} \cup$$
$$\{\langle e \to l''_i; l''_i \leftrightarrow e\rangle,\ \langle e \to l''_{i+1}; l''_i \leftrightarrow e\rangle, \mid 9 \le i \le 20\}.$$

Now, for each instruction $l_i : (\texttt{ADD}(r), l_{i,1}, l_{i,2}),\ 0 \leq i \leq 8$, we have the following programs

$$P_i = \{\langle l_i \to l_i'; e \leftrightarrow e\rangle,\ \langle e \to a_r; l_i' \leftrightarrow l_i'\rangle,\ \langle l_i' \to l_{i,1}; a_r \leftrightarrow e\rangle,\\ \langle l_i' \to l_{i,2}; a_r \leftrightarrow e\rangle\}.$$

It is easy to see that first label l_i is changed for symbol l_i' inside the cell (the first program given above in P_i), and then symbol e is changed for a_r (by $\langle e \to a_r; l_i' \leftrightarrow l_i'\rangle$, the second program in P_i). Then a_r is sent out to the environment and either the computation continues with simulating the instruction labeled by $l_{i,1}$ or it continues with simulating the instruction labeled by $l_{i,2}$.

For the instructions $l_i : (\texttt{CHECKSUB}(r), l_{i,1}, l_{i,2}),\ 10 \leq i \leq 21$, we have the programs

$$P_i = \{\langle l_i \to l_i'; e \leftrightarrow a_r/e \leftrightarrow e\rangle, \langle a_r \to e; l_i' \leftrightarrow l_i''\rangle, \langle l_i'' \to l_{i,1}; e \leftrightarrow e\rangle,\\ \langle l_i' \to l_{i,2}; e \leftrightarrow e\rangle\}.$$

First, label l_i is changed for symbol l_i' and if the environment contains at least one copy of a_r, then it is brought inside the cell. These changes are performed by the first program listed in P_i. In the case of successfully bringing a_r in the cell, the computation continues with applying the second and the third program (i.e., by $\langle a_r \to e; l_i' \leftrightarrow l_i''\rangle$ and $\langle l_i'' \to l_{i,1}; e \leftrightarrow e\rangle$) , which results in erasing a_r from the cell (rewriting it onto e) and introducing the label of the next instruction to be simulated, $l_{i,1}$. If a_r was not present in the environment, then the fourth program changes l_i' to $l_{i,2}$, the label of the next instruction to be simulated.

For the instruction $l_9 : (\texttt{CHECK}(r), l_{9,1}, l_{9,2})$ we also add these four programs, but with the second and the third one modified to avoid decreasing the value stored in the register. Thus,

$$P_9 = \{\langle l_9 \to l_9'; e \leftrightarrow a_r/e \leftrightarrow e\rangle, \langle a_r \to a_r; l_9' \leftrightarrow l_9''\rangle, \langle l_9'' \to l_{9,1}; a_r \leftrightarrow e\rangle,\\ \langle l_9' \to l_{9,2}; e \leftrightarrow e\rangle\}.$$

Moreover, in order to ensure that the cell exchanges primed objects from inside with double primed objects from the environment, we also consider the programs

$$P_{trap} = \{\langle l_i' \to t; e \leftrightarrow e\rangle,\ \langle l_i' \to t; a_r \leftrightarrow e\rangle,\ \langle t \to t; e \leftrightarrow e\rangle \mid 9 \leq i \leq 21\}$$

where t is a trap-object. If an exchange cannot be realized because the number of double primed symbols produced in the initial phase is insufficient, then the trap-object is introduced and the program $\langle t \to t; e \leftrightarrow e\rangle$ works forever, preventing the halting of the computation.

Considering the P colony above with $C = (O, P)$ with $P = \bigcup_{i=0}^{21} P_i \cup P_{ini} \cup P_{trap}$, we need 27 programs for the initial phase, 36 programs for the ADD instructions, 48 programs for the CHECKSUB instructions, 4 programs for the CHECK instruction, and 27 programs in P_{trap}. This gives us $27 + 36 + 4 + 48 + 27 = 142$, thus our statement is proved. □

3.2 P Colonies Without Checking Rules

Now we show how the use of checking rules can be avoided. First we consider the case of two-objects P colonies with restricted and with unrestricted programs.

Theorem 5

$$IPCol\_2(35,8) = IPCol\_2R(57,8) = \mathbb{N}RE.$$

Proof. We show how the universal register machines U_1 and U_3 from Theorem 1 can be simulated. Let U_3 have nine `ADD` instructions labeled by $l_0, \ldots, l_8$, one `CHECK` instruction labeled by l_9, and twelve `CHECKSUB` instructions labeled by $l_{10}, \ldots, l_{21}$, as in the previous theorem.

Consider $\Pi_3 = (V, e, a_1, I_E, C_0, \ldots, C_{21})$ from the proof of Theorem 2 and let $\Pi'_3 = (V', e, a_1, I_E, C'_0, \ldots C'_{34})$ where $V' = V \cup \{l'_i, l''_i \mid 0 \leq i \leq 21\}$. Now we construct the cells $C'_i,\ 0 \leq i \leq 34$, which achieve the same effect as the 21 cells of Π_3. Note that the `ADD` instructions are simulated in Π_3 by $P_i,\ 0 \leq i \leq 8$, without checking rules, thus we can take

$$C'_i = C_i,\ 0 \leq i \leq 8,$$

without any change at all.

For the instruction $l_9 : (\texttt{CHECK}(r), l_{9,1}, l_{9,2})$ we need two cells in Π'_3, having at most eight restricted programs as follows.

$$\begin{aligned} P'_9 = \{&\langle e \to l'_{9,2}; e \leftrightarrow l_9\rangle,\ \langle l_9 \to l'_9; l'_{9,2} \leftrightarrow e\rangle,\ \langle l'_9 \to l''_9; e \leftrightarrow a_r\rangle,\\ &\langle l''_9 \to l_{9,1}; a_r \leftrightarrow e\rangle,\ \langle l_{9,1} \to l_{9,1}; e \leftrightarrow l''_{9,2}\rangle,\ \langle l'_9 \to l_{9,2}; e \leftrightarrow l''_{9,2}\rangle,\\ &\langle l''_{9,2} \to e; l_{9,1} \leftrightarrow e\rangle,\ \langle l''_{9,2} \to e; l_{9,2} \leftrightarrow e\rangle\}, \end{aligned}$$

and

$$P'_{22} = \{\langle e \to l''_{9,2}; e \leftrightarrow l'_{9,2}\rangle,\ \langle l'_{9,2} \to e; l''_{9,2} \leftrightarrow e\rangle\}.$$

The interplay of these two cells produces the desired "checking" effect, they exchange the symbol l_9 to $l_{9,1}$ if there is at least one a_r symbol in the environment, otherwise $l_{9,2}$ is released. In more details, first cell C'_9 imports label l_9 from the environment and meantime introduces symbol $l'_{9,2}$ (the first program given in P'_9). Then, by program $\langle l_9 \to l'_9; l'_{9,2} \leftrightarrow e\rangle$, it changes l_9 to l'_9 and exports $l'_{9,2}$ to the environment. In the next step cell C'_{22} starts working, the activity of C'_9 depends on whether or not the environment contains at least one occurrence of a_r. In the next two steps C'_{22} performs its programs which results in eliminating $l'_{9,2}$ from and exporting $l''_{9,2}$ to the environment. After then, C'_{22} stops with its activity. If the environment contains at least one copy of a_r, then meantime C'_9 brings one a_r inside the cell and rewrites l'_9 onto l''_9 (the third program of P'_9, i.e., $\langle l'_9 \to l''_9; e \leftrightarrow a_r\rangle$). The process continues with performing the fourth program in P'_9, by returning a_r to the environment and changing l''_9 to $l_{9,1}$. In the next two steps, since the environment contains $l''_{9,2}$, cell C'_9, by performing the fifth and seventh program in P'_9, above, eliminates $l''_{9,2}$ from the environment and sends

the label of the next instruction to be simulated, $l_{9,1}$ out from the cell. If the environment does not contain a_r, then after performing the second program, i.e., $\langle l_9 \to l'_9; l'_{9,2} \leftrightarrow e\rangle$, cell C'_9 must stop with its activity. It starts working again after C'_{22} had already sent $l''_{9,2}$ to the environment. By applying the sixth and eights program in P'_9, C'_9 brings $l''_{9,2}$ from the environment and sends out $l_{9,2}$, the label of the next instruction to be simulated.

For the instructions $l_i : (\texttt{CHECKSUB}(r), l_{i,1}, l_{i,2}),\ 10 \leq i \leq 21$, we also need two cells with at most eight programs as follows. Let for each $i,\ 10 \leq i \leq 21$,

$$\begin{aligned} P'_i = \{&\langle e \to l'_{i,2}; e \leftrightarrow l_i\rangle,\ \langle l_i \to l'_i; l'_{i,2} \leftrightarrow e\rangle,\ \langle l'_i \to l''_i; e \leftrightarrow a_r\rangle,\\ &\langle a_r \to e; l''_i \to l_{i,1}\rangle,\ \langle l_{i,1} \to l_{i,1}; e \leftrightarrow l''_{i,2}\rangle,\ \langle l'_i \to l_{i,2}; e \leftrightarrow l''_{i,2}\rangle,\\ &\langle l''_{i,2} \to e; l_{i,1} \leftrightarrow e\rangle,\ \langle l''_{i,2} \to e; l_{i,2} \leftrightarrow e\rangle\}, \end{aligned}$$

and

$$P'_{13+i} = \{\langle e \to l''_{i,2}; e \leftrightarrow l'_{i,2}\rangle,\ \langle l'_{i,2} \to e; l''_{i,2} \leftrightarrow e\rangle\}.$$

The reader can easily observe that these pairs work together almost the same manner as the ones above. The only difference is that, they not only check, but if possible, also decrease the number of a_r symbols in the environment.

Thus, if we have

$$C'_i = (ee, P'_i),\ 9 \leq i \leq 34,$$

in Π'_3, then we can simulate U_3 with 35 cells, each having at most eight non-restricted programs, but no checking rules.

For the proof of the statement concerning two-objects P colonies with restricted rules, we simulate the universal machine U_1 from Theorem 1. U_1 has nine `ADD` instructions labeled by $l_0, \ldots, l_8$, thirteen `CHECK` instructions labeled by $l_9, \ldots, l_{21}$, and eleven `SUB` instructions labeled by $l_{22}, \ldots, l_{32}$.

Let $\Pi''_3 = (V'', e, a_1, I_E, C''_0, \ldots C''_{56})$ with $V'' = \{e\} \cup \{l_i, l'_i, l''_i \mid 0 \leq i \leq 32\} \cup \{a_j \mid 1 \leq j \leq 8\}$, and let

$$C''_i = C'_i,\ 0 \leq i \leq 8,\ \text{where } C'_i \text{ is as above.}$$

For each instruction $l_i : (\texttt{CHECK}(r), l_{i,1}, l_{i,2}),\ 9 \leq i \leq 21$, we need two cells C''_i and C''_{24+i}, as described above for P'_9 and P'_{22}.

For each instruction $l_i : (\texttt{SUB}(r), l_{i,1}),\ 22 \leq i \leq 32$, we need two cells in the P colony, having at most seven restricted programs as follows. For $i,\ 22 \leq i \leq 32$, let

$$\begin{aligned} P_i = \{&\langle e \to l'_{i,1}; e \leftrightarrow l_i\rangle,\ \langle l_i \to l'_i; l'_{i,1} \leftrightarrow e\rangle,\ \langle l'_i \to l''_i; e \leftrightarrow a_r\rangle,\\ &\langle l''_i \to l_{i,1}; a_r \to e\rangle,\ \langle l_{i,1} \to l_{i,1}; e \leftrightarrow l''_{i,1}\rangle,\ \langle l'_i \to l_{i,1}; e \leftrightarrow l''_{i,1}\rangle,\\ &\langle l''_{i,1} \to e; l_{i,1} \leftrightarrow e\rangle\}, \end{aligned}$$

and

$$P_{24+i} = \{\langle e \to l''_{i,1}; e \leftrightarrow l'_{i,1}\rangle,\ \langle l'_{i,1} \to e; l''_{i,1} \leftrightarrow e\rangle\}.$$

The interplay of these pairs results in the decrease of the number of symbols a_r in all cases when this number is non-zero. It is easy to see that the two cells work in a similar manner as the two cells that are presented above which simulate instruction $l_i : (\texttt{CHECKSUB}(r), l_{i,1}, l_{i,2})$. The only difference is that both in the case of presence of a_r in the environment and in the case of its absence, the computation continues with simulating the instruction labeled by $l_{i,1}$.

Now, if we have

$$C''_i = (ee, P''_i),\ 9 \leq i \leq 56,$$

in Π''_3, then we can simulate U_1 with 57 cells, each having at most seven restricted program, without checking rules. □

The number of programs in the cells can be decreased if, instead of two, we allow three objects in each cell, and thus, three instructions in each program.

Theorem 6

$$IPCol\_3(35, 7) = \mathbb{N}RE.$$

Proof. Consider $\Pi'_3 = (V', e, a_1, I_E, C'_0, \ldots C'_{34})$ simulating U_3 from Theorem 1 as described in the first part of the proof of the previous theorem. Now define $C''_i = (eee, P''_i),\ 0 \leq i \leq 35$, where for $i,\ 0 \leq i \leq 8$, P''_i is obtained from P'_i by adding a rule $e \to e$ to each program.

For $l_9 : (\texttt{CHECK}(r), l_{9,1}, l_{9,2})$, we have

$$\begin{aligned} P''_9 = \{&\langle e \to l'_{9,2}; e \to l'_{9,1}; e \leftrightarrow l_9\rangle,\ \langle l_9 \to e; l'_{9,2} \leftrightarrow e; l'_{9,1} \to l'_{9,1}\rangle,\\ &\langle l'_{9,1} \to l''_{9,1}; e \leftrightarrow a_r; e \leftrightarrow e\rangle,\ \langle a_r \leftrightarrow e; l''_{9,1} \to l_{9,1}; e \leftrightarrow l''_{9,2}\rangle,\\ &\langle l'_{9,1} \to e; e \leftrightarrow l''_{9,2}; e \to l_{9,2}\rangle,\ \langle l''_{9,2} \to e; l_{9,1} \leftrightarrow e, e \to e\rangle,\\ &\langle l''_{9,2} \to e; l_{9,2} \leftrightarrow e; e \to e\rangle\}, \end{aligned}$$

and

$$P''_{22} = \{\langle e \to l''_{9,2}; e \leftrightarrow l'_{9,2}; e \to e\rangle,\ \langle l'_{9,2} \to e; l''_{9,2} \leftrightarrow e; e \to e\rangle\},$$

The interplay of the two cells is very similar to the interplay of cell pairs constructed to simulate the corresponding CHECK instructions in the previous proofs. The simulation starts with the work of cell C''_9. By applying the first program given in P''_9 symbol l_9 is brought in the cell and the two remaining symbols e are rewritten onto symbols $l'_{9,2}$ and $l'_{9,1}$. At the next step, the cell sends out $l'_{9,2}$ to the environment, l_9 is rewritten to e and $l'_{9,1}$ remains unchanged. At the following step, C''_{22} imports $l'_{9,2}$ from the environment and rewrites one occurrence of e onto $l''_{9,2}$. Meantime C''_9 either imports one a_r from the environment and rewrites $l'_{9,1}$ onto $l''_{9,1}$ or, if there is no occurrence of a_r in the environment, it stops with its activity. In the following step, C''_{22} sends $l''_{9,2}$ to the environment and rewrites $l'_{9,2}$ to e and finishes its activity. During this step C''_9 does not work, it has no program to be applied. After the appearance of $l''_{9,2}$ in the environment, C''_9 starts working again. Then either by applying the fourth and the sixth program or the fifth and the seventh program given above in P''_9 it finishes the simulation by

exporting $l_{9,1}$ or $l_{9,2}$ to the environment, respectively. In the first case, i.e., when the next instruction to be simulated is labeled by $l_{9,1}$, C''_9 returns the copy of a_r to the environment, thus verifies the presence of a_r in the environment.

For the rules $l_i : (\texttt{CHECKSUB}(r), l_{i,1}, l_{i,2}),\ 10 \leq i \leq 21$, let us define

$$\begin{aligned} P''_i = \{&\langle e \to l'_{i,2}; e \to l'_{i,1}; e \leftrightarrow l_i\rangle,\ \langle l_i \to e; l'_{i,2} \leftrightarrow e; l'_{i,1} \to l'_{i,1}\rangle, \\ &\langle l'_{i,1} \to l''_{i,1}; e \leftrightarrow a_r; e \to e\rangle,\ \langle a_r \to e; l''_{i,1} \to l_{i,1}; e \leftrightarrow l''_{i,2}\rangle, \\ &\langle l'_{i,1} \to e; e \leftrightarrow l''_{i,2}; e \to l_{i,2}\rangle,\ \langle l''_{i,2} \to e; l_{i,1} \leftrightarrow e, e \to e\rangle, \\ &\langle l''_{i,2} \to e; l_{i,2} \leftrightarrow e; e \to e\rangle\}, \end{aligned}$$

and

$$P''_{13+i} = \{\langle e \to l''_{i,2}; e \leftrightarrow l'_{i,2}; e \to e\rangle,\ \langle l'_{i,2} \to e; l''_{i,2} \leftrightarrow e; e \to e\rangle\}.$$

These cells work together very similarly as the cell pair C''_9 and C''_{22}, above, the only difference is that C''_i not only checks the presence/absence of symbol a_r in the environment but if the environment has at least one occurrence of a_r, then it eliminates one copy of a_r from the environment. Thus, if we consider the P colony $\Pi''_3 = (V', e, a_1, I_E, C''_0, \ldots C''_{34})$, then our statement is proved. □

4 Conclusion

We have shown that P colonies are able to simulate universal register machines, provided they are initialized as follows: besides the infinitely many copies of the environmental object, a finite number of other objects are placed in the environment. Thus, P colonies are able to generate any recursively enumerable set of non-negative integers with a bounded number of cells, each containing a bounded number of programs of a bounded length.

We have considered different types of universal machines and different types of P colonies, but still, our results are only a first approximation of the number of necessary cells and programs. To decrease the present bounds, or to prove that they are sharp ones, remains a topic of further research.

References

1. Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, Gh.: *Grammar Systems – A Grammatical Approach to Distribution and Cooperation*. Gordon and Breach, London, 1994.
2. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., Păun, Gh., Vaszil, Gy.: Computing with cells in environment: P colonies. *Journal of Multiple Valued Logic and Soft Computing* (to appear).
3. Kelemen, J., Kelemenová, A.: A grammar-theoretic treatment of multi-agent systems. *Cybernetics and Systems*, **23** (1992), 621–633.
4. Kelemen, J., Kelemenová, A., Păun, Gh.: Preview of P colonies: A biochemically inspired computing model. In: *Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems* (Alife IX) (M. Bedau et al., eds.), Boston Mass., 2004, 82–86.

5. Korec, I.: Small universal register machines. *Theoretical Computer Science*, **168** (1996), 267–301.
6. Minsky, M.: *Computation – Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, NJ, 1967.
7. Păun, Gh.: *Membrane Computing – An Introduction.* Springer-Verlag, Berlin, 2002.

P Finite Automata and Regular Languages over Countably Infinite Alphabets*

Jürgen Dassow[1] and György Vaszil[2]

[1] Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
PSF 4120, D-39016 Magdeburg, Germany
dassow@iws.cs.uni-magdeburg.de
[2] Computer and Automation Research Institute
Hungarian Academy of Sciences
Kende utca 13-17, 1111 Budapest, Hungary
vaszil@sztaki.hu

Abstract. We examine the class of languages over countably infinite alphabets characterized by a class of restricted and simplified P automata variants, which we call P finite automata, and show that these classes possess several properties which make them perfect candidates for being the natural extension of the notion of finite automata and that of regular languages to infinite alphabets. To this aim, we also show that P finite automata are equivalent to a restricted variant of register machines, providing a more conventional automata theoretical characterization of the same infinite alphabet language class.

1 Extending the Chomsky Hierarchy to Infinite Alphabets

We wish to use the framework of P automata to combine two approaches used earlier to handle languages over infinite alphabets with devices having a finite description, that is, with devices which can be described without the necessity of specifying an infinite set of transition rules, and to define in some reasonable way the infinite alphabet counterparts of classical language classes from the Chomsky hierarchy.

One of these approaches can be summarized as enabling the device to remember a finite number of symbols from the infinite alphabet. If we think of machines accepting strings of symbols, they might be equipped with a certain kind of data structure made of memory slots capable of storing arbitrary symbols of the infinite alphabet, and with the ability to make equality checks between the input symbols and some parts of the memory contents. Then, if we also create rules to manipulate the contents of the memory, the transitions can be given by reference to a finite subset of the memory slots and not to the input symbols themselves.

* Research supported in part by the Hungarian Scientific Research Fund "OTKA" grant no. F037567 and by the Alexander von Humboldt Foundation of the Federal Republic of Germany.

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 367–381, 2006.

An example of this approach is [3] where the authors extend the notion of regular languages to infinite alphabets by defining them as sets of strings accepted by so called finite memory automata, finite automata equipped with a finite set of memory registers capable of storing symbols of the input. The transitions which can also manipulate in some simple way the contents of the memory, are based on the internal state of the finite control unit, and the equality (or non-equality) of the input symbol with the contents of certain registers. A similar idea is used in [1] to extend the notion of pushdown automata and of context-free grammars to infinite alphabets. Since equality check in this framework is "easy", the language $\{a_i^{2n} \mid i, n \geq 1\}$ over the alphabet $\Sigma = \{a_i \mid i \geq 1\}$, for example, can easily be accepted by finite memory automata. On the other hand, to capture relationships of the symbols other than equality or non-equality is "hard", the language $\{a_{2i} \mid i \geq 1\}$ over Σ, for example, cannot be characterized with any of the above mentioned devices.

The second approach for handling infinite alphabets can be summarized as coding the symbols using a finite alphabet and then working with the corresponding code-word language (over a finite alphabet) instead of the original language (which is over an infinite alphabet).

As an example, we could mention [5] where a symbol a_i from the infinite alphabet $\Sigma = \{a_i \mid i \geq 1\}$ is coded as the word 0^i1 over the binary alphabet $\{0, 1\}$, and then a language over the infinite alphabet Σ is defined to be regular (or context-free), if the corresponding code-word language over the binary alphabet $\{0, 1\}$ is, in the conventional sense, regular (or context-free). This approach has some advantages, the language $\{a_{2i} \mid i \geq 1\}$, for example, which cannot be characterized with finite memory automata, is clearly regular according to this definition. On the other hand, the equality check of symbols proves to be "hard" in this framework, the language $\{a_i a_i \mid i \geq 1\}$ for example, is in this sense a non-regular, or $\{a_i a_i a_i \mid i \geq 1\}$ is a non-context-free language.

In the present paper, we propose the combination of the previously described approaches: Instead of just coding or just remembering symbols, we propose to remember the codes of symbols. In some sense, this can not only be seen as a proposal which eliminates certain shortcomings of the above outlined concepts, but also as a more "realistic" description of the act of remembering a symbol from an infinite alphabet: The remembered object needs to be somehow denoted, and for this, a finite collection of signs, elements of a notation, can be used, thus, when we think of storing symbols, we really mean to think of storing code-words corresponding to symbols, and exactly this fact is expressed in our proposed model.

To explore these ideas, we will use the framework of membrane systems, or more precisely of P automata, which provide a very natural machinery to capture the above described concepts. The introduction of P automata in [2] was motivated by the idea of using P systems as language acceptors while keeping the machinery as simple as possible. The objects in a P automaton may move through the membranes from region to region, but they may not be modified during the functioning of the systems, and furthermore, the "words" accepted by

a P automaton correspond to the sequences of multisets containing the objects entering from the environment in each step of the evolution of the system.

Although the number of different objects used by the system, that is, the number of elements of the object alphabet of the system is finite, the number of possible multisets of objects entering the skin membrane in one computational step can be infinite. The reason for this property lies in the parallelism of the application of the evolution rules (which are communication rules in this case). The number of objects manipulated by one rule is finite, but since they can be applied in any number of "copies", the number of objects affected by the rules in one computational step can be arbitrary high, thus the potential number of objects requested by the rules of the skin membrane to enter the system is unbounded. Because of the infinite number of potential input multisets, it is rather natural to consider a P automaton as a machine working with strings of symbols over infinite alphabets.

In the following we define a quite restrictive class of such P automata, which we call P finite automata, to obtain a reasonably simple, but on the other hand, a still rather complex class of languages over countably infinite alphabets, and examine how appropriate candidate this language class is for being called the "infinite alphabet counterpart" of regular languages. To explore the relationship of our P automaton based model and the ones based on more conventional type of automata, we also define restricted variants of register machines which have additional capabilities for dealing with countably infinite alphabets, but a very restricted way of using the registers, the motivation of these restrictions being to formalize the properties of the P automaton based model in a more conventional-like automata theoretical framework.

We will show that the languages over finite alphabets contained by the language class accepted by P finite automata are precisely the regular languages, thus, since all infinite alphabet languages mentioned above as examples are also contained in this class, with P finite automata we can obtain a more adequate generalization of finite automata, and of regular languages, to infinite alphabets, then with any of the above mentioned approaches.

2 Preliminaries and Definitions

We first recall the notions and the notations we use. The reader is assumed to be familiar with the basics of formal language theory, for details see [8]. Let Σ be a not necessarily finite, but countable set of symbols called alphabet. Let Σ^* be the set of all words over Σ, that is, the set of finite strings of symbols from Σ, and let $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ where ε denotes the empty word.

Let V be a set of objects, let $\mathbb{N}$ and $\mathbb{Z}$ denote the set of nonnegative integers and the set of integers respectively. A multiset is a mapping $M : V \to \mathbb{N}$ which assigns to each object $a \in V$ its multiplicity $M(a)$ in M. The support of M is the set $supp(M) = \{a \mid M(a) \geq 1\}$. If $supp(M)$ is a finite set, then M is called a finite multiset. The set of all finite multisets over the set V is denoted by V°.

We say that $a \in M$ if $M(a) \geq 1$. For $M_1, M_2 : V \to \mathbb{N}$, the containment relation $M_1 \subseteq M_2$ holds if for all $a \in V$, $M_1(a) \leq M_2(a)$. The union of M_1 and

M_2 is defined as $(M_1 \cup M_2) : V \to \mathbb{N}$ with $(M_1 \cup M_2)(a) = M_1(a) + M_2(a)$ for all $a \in V$, the difference is defined for $M_2 \subseteq M_1$ as $(M_1 - M_2) : V \to \mathbb{N}$ with $(M_1 - M_2)(a) = M_1(a) - M_2(a)$ for all $a \in V$, and the intersection is $(M_1 \cap M_2) : V \to \mathbb{N}$ with $(M_1 \cap M_2)(a) = min(M_1(a), M_2(a))$ for $a \in V$, where $min(x, y)$ denotes the minimum of $x, y \in \mathbb{N}$. We say that M is empty if its support is empty, $supp(M) = \emptyset$.

A multiset M over the finite set of objects V can be represented as a string w over the alphabet V with $|w|_a = M(a)$ where $a \in V$ and $|w|_a$ denotes the number of occurrences of the symbol a in the string w, and with ε representing the empty multiset. In the following we sometimes identify the finite multiset of objects $M : V \to \mathbb{N}$ with the word w over V representing M, thus we write $w \in V^\circ$, or sometimes we enumerate the not necessarily distinct elements $a_1, \ldots, a_n$ of a multiset as $M = \{\{a_1, \ldots, a_n\}\}$, by using double brackets to distinguish from the usual set notation.

A P system is a structure of hierarchically embedded membranes, each having a label and enclosing a region containing a multiset of objects and possibly other membranes. The out-most membrane which is unique and usually labeled with 1, is called the skin membrane. The membrane structure is denoted by a sequence of matching parentheses where the matching pairs have the same label as the membranes they represent. If membrane i contains membrane j, and there is no other membrane, k, such that k contains j and i contains k, then we say that membrane i is the parent membrane of j, denoted as $parent(j) = i$.

The evolution of the contents of the regions of a P system is described by rules associated to the regions. Applying the rules synchronously in each region, the system performs a computation by passing from one configuration to another one. Several variants of the basic notion have been introduced and studied proving the power of the framework, see the monograph [7] for a summary of notions and results of the area. In the following we concentrate on communication rules called symport or antiport rules.

A symport rule is of the form (x, in) or $(x, out), x \in V^\circ$. If such a rule is present in a region i, then the objects of the multiset x can enter from the parent region or can leave to the parent region, respectively. An antiport rule is of the form $(x, in; y, out), x, y \in V^\circ$, in this case, objects of x enter from the parent region and in the same step, objects of y leave to the parent region. All types of these rules might be equipped with a promoter or inhibitor multiset, denoted as $(x, in)|_Z, (x, out)|_Z$, or $(x, in; y, out)|_Z$, with $x, y \in V^\circ, Z \in \{z, \neg z \mid z \in V^\circ\}$, where if $Z = z$ then the rules can only be applied if region i contains the objects of multiset z, or if $Z = \neg z$, then region i must not contain any of the elements of z. (For more on symport/antiport see [6], for the use of promoters see [4].)

A *P automaton* is $\Gamma = (V, \mu, (w_1, P_1, F_1), \ldots, (w_n, P_n, F_n))$ where $n \geq 1$ is the number of membranes, V is a finite set of objects, μ is a membrane structure of n membranes with membrane 1 being the skin membrane, and for all $i, 1 \leq i \leq n$,

- $w_i \in V^\circ$ is the initial contents (state) of region i, that is, it is the finite multiset of all objects contained initially by region i,

- P_i is a finite set of communication rules associated to membrane i, they can be symport rules or antiport rules, with or without promoters or inhibitors, as above, and
- $F_i \subseteq V^\circ$ is a finite set of finite multisets over V called the set of final states of region i. If $F_i = \emptyset$, then all the states of membrane i are considered to be final.

To simplify the notations we denote symport and antiport rules with or without promoters/inhibitors as $(x, in; y, out)|_Z$, $x, y \in V^\circ$, $Z \in \{z, \neg z \mid z \in V^\circ\}$, thus we also allow x, y, z to be the empty multiset/empty string. If $y = \varepsilon$ or $x = \varepsilon$, then the notation above denotes the symport rule $(x, in)|_Z$ or $(y, out)|_Z$, respectively, if $z = \varepsilon$, then the rules above are without promoters or inhibitors, they can also be denoted as $(x, in; y, out)$. We might also exchange the order of multisets to be sent out and to be imported, that is, the rule $(x, out; y, in)|_Z$ is equivalent to $(y, in; x, out)|_Z$, with x, y, Z as above.

The n-tuple of finite multisets of objects present in the n regions of the P automaton Γ describes a *configuration* of Γ, the n-tuple $(w_1, \ldots, w_n) \in (V^\circ)^n$ is the initial configuration.

We say that a promoter or inhibitor Z of a rule $(x, in; y, out)|_Z \in P_i$ is consistent with a configuration $(w_1, \ldots, w_n)$, if it permits the application of the rule, that is, either $Z = z \in V^\circ$ and $z \subseteq w_i$, or $Z = \neg z,\ z \in V^\circ$ and $z \cap w_i = \varepsilon$.

The transition mapping of a P automaton is a partial mapping $\delta : V^\circ \times (V^\circ)^n \to 2^{(V^\circ)^n}$. These mappings are defined implicitly by the rules of the sets $P_i,\ 1 \leq i \leq n$. For a configuration $(u_1, \ldots, u_n)$,

$$(u'_1, \ldots, u'_n) \in \delta(u, (u_1, \ldots, u_n))$$

holds, that is, while reading the input $u \in V^\circ$ the automaton may enter the new configuration $(u'_1, \ldots, u'_n) \in (V^\circ)^n$, if there exist rules as follows.

- For all $i, 1 \leq i \leq n$, there is a multiset of rules $R_i = \{\{r_{i,1}, \ldots, r_{i,m_i}\}\}$, where $r_{i,j} = (x_{i,j}, in; y_{i,j}, out)|_{Z_{i,j}} \in P_i$ and $Z_{i,j},\ 1 \leq j \leq m_i$, is consistent with u_i, satisfying the conditions below, where $x_i,\ y_i$ denote the multisets $\bigcup_{1 \leq j \leq m_i} x_{i,j}$ and $\bigcup_{1 \leq j \leq m_i} y_{i,j}$, respectively. Furthermore, there is no rule occurrence $r \in P_j$, being consistent with u_j, $1 \leq j \leq n$, with the additional restriction that if $r \in P_1$ then $r \neq (x, in)|_Z$, such that the rule multisets R'_i with $R'_i = R_i$ for $i \neq j$ and $R'_j = \{\{r\}\} \cup R_j$, also satisfy the conditions.

The conditions are given as

1. $x_1 = u$, and
2. $(\bigcup_{parent(j)=i} x_j) \cup y_i \subseteq u_i,\ 1 \leq i \leq n$,

and then the new configuration is obtained by

$$u'_i = u_i \cup x_i - y_i \cup \bigcup_{parent(j)=i} y_j - \bigcup_{parent(j)=i} x_j,\ 1 \leq i \leq n.$$

Note that we allow the use of rules of type $(x, in)|_Z$ in the skin membrane in such a way that the application of any number of copies of these rules is considered to be maximally parallel.

Let us extend δ to $\bar{\delta}$, a function mapping $(V^\circ)^*$, the sequences of finite multisets over V, and $(V^\circ)^n$, the configurations of Γ, to new configurations.

1. $\bar{\delta}(v, (u_1, \ldots, u_n)) = \delta(v, (u_1, \ldots, u_n)),\ v, u_i \in V^\circ,\ 1 \leq i \leq n$, and
2. $\bar{\delta}((v_1) \ldots (v_{s+1}), (u_1, \ldots, u_n)) = \bigcup \delta(v_{s+1}, (u'_1, \ldots, u'_n))$ for all $(u'_1, \ldots, u'_n) \in \bar{\delta}((v_1) \ldots (v_s), (u_1, \ldots, u_n)),\ v_j, u_i, u'_i \in V^\circ,$ $1 \leq i \leq n,\ 1 \leq j \leq s+1$.

Note that we use brackets in the multiset sequence $(v_1) \ldots (v_{s+1}) \in (V^\circ)^*$ in order to distinguish it from the multiset $v_1 \cup \ldots \cup v_{s+1} \in V^\circ$.

The sequence of multisets of objects accepted by the P automaton is the sequence of input multisets consumed by the skin membrane during the sequence of computational steps while the system reaches a final state, a configuration where for all j with $F_j \neq \emptyset$, the contents $u_j \in V^\circ$ of membrane j is "final", i.e., $u_j \in F_j$.

Let Γ be a P automaton as above with initial configuration $(w_1, \ldots, w_n)$, let Σ be an alphabet, and let $f : V^\circ \longrightarrow \Sigma \cup \{\varepsilon\}$ be a mapping with $f(x) = \varepsilon$ if and only if $x = \varepsilon$.

We obtain the words of the *language accepted by* Γ as the images of the accepted multiset sequences, that is,

$$L(\Gamma, f) = \{f(v_1) \ldots f(v_s) \in \Sigma^* \mid (u_1, \ldots, u_n) \in \bar{\delta}((v_1) \ldots (v_s), (w_1, \ldots, w_n)) \text{ with } u_j \in F_j \text{ for all } j \text{ with } F_j \neq \emptyset,\ 1 \leq j \leq n,\ 1 \leq s\}.$$

Obviously, the choice of the mapping f is essential. It has to be "easily" computable because the power of the P automaton should be provided by the underlying membrane system and not by f itself. The notion of "easiness", however, greatly depends on the context we are working in, so we do not give it a general specification here.

3 P Finite Automata and Restricted Register Automata

Now we introduce restricted variants of P automata which we call P finite automata. The object alphabet of a P finite automaton contains a distinct element which is the only one that can appear in an arbitrary number of copies inside the membrane structure and can be present only in the skin membrane. The other objects can move around through the regions, but they can only be exported, thus, their number of occurrences cannot increase during any computation. These properties are ensured by the very special and very simple form of rules which can be used by the system.

Definition 1. A *P finite automaton* (PFA in short) is a P automaton $(V \cup \{a\}, \mu, (w_1, P_1, F_1), \ldots, (w_n, P_n, F_n))$ where $V \cup \{a\}$ is a finite alphabet with a

distinct element denoted by a, μ is a membrane structure of n membranes, and for $1 \leq i \leq n$, $w_i \in V^\circ$ is the initial multiset of region i, F_i is the set of final states for region i, and P_i is a finite set of rules associated to region i where

- if $i \neq 1$, P_i contains rules of the form $(x, in; y, out)|_Z$ with $Z \in \{z, \neg z\}$, $x, y, z \in V^\circ$, and
- P_1 contains rules of the form $(x, in; y, out)|_Z$ where $x \in \{a\}^\circ$, $y \in (V \cup \{a\})^\circ$, $Z \in \{z, \neg z\}$, $z \in V^\circ$.

Thus, a PFA can only input multisets of the symbol a from the environment and these symbols can only remain in the skin region or be sent back to the environment, but they cannot enter regions i with $i \geq 2$. The symbol a is the only one which can appear in arbitrary many copies inside the system, the number of the other letters is at most as many as in the initial configuration, it might only decrease during the computation.

The correspondence between the set of possible input multisets and the countably infinite alphabet $\Sigma = \{a_i \mid i \geq 1\}$ is based on the number of a symbols in the input. If $M \in \{a\}^\circ$ is an input multiset containing k copies of the symbol a, that is, $M(a) = k$, then M is mapped to $a_k \in \Sigma$. Thus, a sequence of input multisets corresponds to a sequence of letters from Σ. This is expressed in the following definition.

Definition 2. Let $\Sigma = \{a_i \mid i \geq 1\}$ be a countably infinite alphabet, and let Γ be a PFA as above. The language over Σ accepted by Γ, denoted as $L(\Gamma)$, is defined as follows.

$$L(\Gamma) = \{a_{i_1} a_{i_2} \dots a_{i_s} \in \Sigma^* \mid (u_1, \dots, u_n) \in \bar{\delta}((v_1) \dots (v_s), (w_1, \dots, w_n)) \text{ with } u_j \in F_j \text{ for all } j \text{ with } F_j \neq \emptyset,\ 1 \leq j \leq n, \text{ and } f(v_k) = a_{i_k},\ 1 \leq k \leq s\}$$

where f is defined as $f : \{a\}^\circ \to \Sigma \cup \{\varepsilon\}$, with $f(M) = a_i$ where $M(a) = i$ for a multiset $M(a) > 0$, and $f(M) = \varepsilon$ for M with $M(a) = 0$, that is, if M is empty.

Let $\mathcal{L}(PFA)$ denote the class of languages accepted by P finite automata.

Now we define restricted two register automata, or RRA in short, an other type of machine model to capture the capabilities of P finite automata. As we will show, the two models are equivalent, that is, they characterize the same class of infinite alphabet languages.

An RRA has a finite control unit and two registers holding nonnegative integer values. The machine is capable of changing states, subtracting certain values from the first register and adding other values to the second one, until no more modifications of this type are possible. At this point, if the value of the second register is k, the machine reads an input symbol a_k from the countably infinite input alphabet, empties the second register, and adds its value to the contents of the first register. Then it continues its work by starting again the subtracting and adding process.

Definition 3. A *restricted register finite automaton*, or RRA in short, is a construct $M = (\Sigma, Q, P, q_0, r_0, F)$ where $\Sigma = \{a_i \mid i \geq 1\}$ is a countably infinite alphabet, Q is a finite set of states of the form $Q = \{0, 1, \ldots, c\}^n \times \{0, -1, \ldots, -c\}^n \times \{0, 1, \ldots, c\}^n$ for some positive integer $c \in \mathbb{N}$, the initial state is $q_0 \in \{0, 1, \ldots, c\}^n \times \{0\}^{2n} \subset Q$, $r_0 \in \mathbb{N}$ is a nonnegative integer, the initial contents of the first register, $F \subseteq \{0, 1, \ldots, c\}^n \times \{0\}^{2n} \subset Q$ is the set of final states, P is a finite set of instructions of the form $[q; i_1, i_2]$, or $[q; p_1 p_2 \ldots p_{2n}; i_1, i_2]$ where $q \in \{0, 1, \ldots, c\}^n \times \{0\}^{2n} \subset Q$, $i_1, i_2 \in \mathbb{N}$, and $p_i \in \mathbb{Z}$, $1 \leq i \leq 2n$, with $p_i \leq 0$ for $1 \leq i \leq n$, and $p_i \geq 0$ for $n+1 \leq i \leq 2n$. Furthermore, for $[q; p_1 p_2 \ldots p_{2n}; i_1, i_2] \in P$,

- if $i_1 = i_2 = 0$, then $\sum_{i=1}^{2n} p_i = 0$, or
- if $i_1 + i_2 > 0$, then $p_i = 0$ for $n+1 \leq i \leq 2n$, and $\sum_{i=1}^{2n} p_i < 0$. (1)

We call two states, $q, q' \in Q$, similar, denoted as $q \sim q'$, if the first n coordinates coincide, that is, if $q = (c_1, \ldots, c_{3n})$ and $q' = (c'_1, \ldots, c'_{3n})$ with $c_i = c'_i$ for $1 \leq i \leq n$.

A configuration of M is a triple (q, r_1, r_2) with the current state $q \in Q$, and the current register contents $r_1, r_2 \in \mathbb{N}$. Given a configuration (q, r_1, r_2), the register contents can be modified obtaining (q, r'_1, r'_2), denoted as

$$(q, r_1, r_2) \Rightarrow (q, r'_1, r'_2),$$

if there is an instruction $[q'; i_1, i_2] \in P$ for some $q' \sim q$ with $r_1 - i_1 \geq 0$, $r'_1 = r_1 - i_1$, $r'_2 = r_2 + i_2$. Thus, the instruction can be applied in any $q \in Q$ being similar to q' by subtracting the value of i_1 from the first register, and adding i_2 is to the second register.

The internal state of the machine can be modified, denoted as

$$(q, r_1, r_2) \Rightarrow (q', r'_1, r'_2),$$

if there is an instruction $[q''; p_1 p_2 \ldots p_{2n}; i_1, i_2] \in P$ for some $q'' \sim q$, where $q = (c_1, \ldots, c_{3n})$, $c_i + p_i \geq 0$, $1 \leq i \leq n$, and $r_1 - i_1 \geq 0$. Then, $q' = (c'_1, \ldots, c'_{3n})$ where $c'_i = c_i$ for $1 \leq i \leq n$, $c'_{n+i} = c_{n+i} + p_i$ for $1 \leq i \leq 2n$, and $r'_1 = r_1 - i_1$, $r'_2 = r_2 + i_2$. Thus, when an instruction of this type is used, then after modifying the register contents using the values i_1, i_2, the machine enters a new state q' obtained from q by adding p_i, $1 \leq i \leq n$, to the $n+i$th component of q (note that $p_i \leq 0$), or adding p_{n+i} to the $2n+i$th component of q (where $p_{n+i} \geq 0$).

The machine can read the input symbol $a_j \in \Sigma$, denoted as

$$(q, r_1, j) \Rightarrow^{a_j} (q', r'_1, 0),$$

where $r'_1 = r_1 + j$, that is, if the contents of the second register is j, the symbol a_j can be read from the input, while j is added to the value of the first register and the contents of the second register is changed to 0. The new state $q' = (c'_1, \ldots, c'_{3n})$ is obtained from $q = (c_1, \ldots, c_{3n})$ such that $c'_i = c_i + c_{i+n} + c_{i+2n}$ for $1 \leq i \leq n$, and $c'_i = 0$ for $n+1 \leq i \leq 3n$.

Let us denote the reflexive and transitive closure of $\Rightarrow$ by $\Rightarrow^*$, and let a transition of M be defined as

$$(q, r_1, 0) \vdash^{a_j} (q', r_1', 0),$$

if there is a sequence of rule applications

$$(q, r_1, 0) \Rightarrow^* (q'', r_1'', j) \Rightarrow^{a_j} (q', r_1', 0),$$

where $r_1' = r_1'' + j$, and the following properties hold:

There is no $[\bar{q}; i_1, i_2] \in P$ such that $\bar{q} \sim q$ with $i_1 \leq r_1''$, and there is no $[\bar{q}; p_1 p_2 \dots p_{2n}; i_1', i_2']$ such that $\bar{q} \sim q$ and if $q'' = (c_1'', \dots, c_{3n}'')$ then $p_i + c_i'' \geq 0$ for all $1 \leq i \leq n$, and $i_1' \leq r_1''$.

Note, that if $(q, r_1, 0) \vdash^{a_j} (q', r_1', 0)$ holds and there is a rule of the form $[q''; 0, i_2] \in P$ with $q \sim q''$, then $(q, r_1, 0) \vdash^{a_{j+k \cdot i_2}} (q', r_1' + k \cdot i_2, 0)$ also holds, for any $k \in \mathbb{N}$.

Note also, that if any of i_1 or i_2 in an instruction $[q; p_1 p_2 \dots p_{2n}; i_1, i_2] \in P$ is not zero, and the instruction was used in a transition $(q, r_1, 0) \vdash^{a_j} (q', r_1', 0)$, then it cannot be used in any other transition since, due to the constraint marked with (1) above, after being in state $q' = (c_1', \dots, c_{3n}')$, the internal control can never enter state $q = (c_1, \dots, c_{3n})$ again, because $\Sigma_{i=1}^{3n} c_i' < \Sigma_{i=1}^{3n} c_i$, and the machine has no way to increase the sum of coordinates in the states.

Definition 4. Let M be a RRA as above. The *language* accepted by M is defined as

$$L(M) = \{w = x_1 \dots x_n \in \Sigma^* \mid (q_0, r_0, 0) = C_0 \vdash^{x_1} C_1 \vdash^{x_2} \dots \vdash^{x_{n-1}} C_{n-1} \vdash^{x_n} C_n = (q_f, r, 0) \text{ where } q_f \in F\}.$$

Let the class of languages (over countably infinite alphabets) accepted by RRA be denoted by $\mathcal{L}(RRA)$.

Now we show that P finite automata and restricted register automata are equivalent, that is, they accept the same class of infinite alphabet languages.

Lemma 1. $\mathcal{L}(RRA) = \mathcal{L}(PFA)$.

Proof. We first show that $\mathcal{L}(RRA) \subseteq \mathcal{L}(PFA)$. Let $\Sigma = \{a_i \mid i \geq 1\}$ be a countably infinite alphabet, and let $L \subseteq \Sigma^*$ be a language accepted by the RRA $M = (\Sigma, Q, P, q_0, r_0, F)$ with $Q = \{0, 1, \dots c\}^n \times \{0, -1, \dots -c\}^n \times \{0, 1, \dots c\}^n$, $c \in \mathbb{N}$. We construct a P finite automaton Γ, such that $L(M) = L(\Gamma)$.

Let $\Gamma = (V \cup \{a\}, [\ [\ ]_2\ ]_1, (w_1, P_1, F_1), (w_2, P_2, \emptyset))$ where $V = \{b_i \mid 1 \leq i \leq n\} \cup \{q, q' \mid q \in Q\} \cup \{A, A', A'', B, C\}$. For each $q = (c_1, \dots, c_{3n}) \in Q$, let us denote with $w(q)$ the multiset $w(q) = b_1^{c_1} \dots b_n^{c_n}$. Note that for all $q \sim \bar{q}$, $q, \bar{q} \in Q$, $w(q) = w(\bar{q})$. Now, let

$$w_1 = a^{r_0} w(q_0) AC,$$
$$P_1 = \{(a^{i_1}, out; a^{i_2}, in)|_{q' A''} \mid [q; i_1, i_2] \in P\} \cup$$

$$\{(a^{i_1}b_1^{-p_1}\dots b_n^{-p_n}, out; a^{i_2}, in)|_{q'A''} \mid [q; p_1\dots p_{2n}; i_1, i_2] \in P,$$
$$i_1 + i_2 > 0\} \cup$$
$$\{(b_iC, out)|_q \mid 1 \le i \le n, q \in Q\} \cup \{(x, out)|_B \mid x \in (V - \{BC\})\ \},$$
$$F_1 = BC,$$

$$w_2 = \{\{a_i^{n\cdot c} \mid 1 \le i \le n\}\} \cup \{\{q, q' \mid q \in Q\}\} \cup \{\{A', A'', B\}\},$$
$$P_2 = \{A'q, out; Aw(q), in)|_{A''}, (A''w(q)q', out; A'q, in),$$
$$(A, out; A''q', in) \mid q \in Q\} \cup \{(B, out; A'q_f, in) \mid q_f \in F\} \cup$$
$$\{(b_1^{p_1+n}\dots b_n^{p_{2n}}, out; b_1^{-p_1}\dots b_n^{-p_n}, in)|_{\neg q'} \mid [q; p_1\dots p_{2n}; 0, 0] \in P\}.$$

We claim that Γ accepts the same words as M. To see this, consider the following. A configuration of Γ containing the multiset $w(q)a^{r_1}AC$ in the first region corresponds to a configuration $(q, r_1, 0)$ of M. Γ is able to check which state of M corresponds to the multiset $w(q)$ by using one of its rules $(A'q, out; Aw(q), in)$ in the second region which exchanges $w(q)$ for the symbol q. Now, the rule $(b_iC, out)|_q$ in the first region makes sure that q is really the state which was corresponding to the contents of the first region since if there are any b_i symbols are left, then C is exported to the environment which makes it impossible for Γ to reach a final state. Now q is brought back to the second region and $w(q)$ is sent back to the first region together with the symbols $A''q'$. When A'' and q' is present in the first region, then the rules $(a^{i_1}, out; a^{i_2}, in)|_{q'A''}$ and $(a^{i_1}b_1^{-p_1}\dots b_n^{-p_n}, out; a^{i_2}, in)|_{q'A''}$ can be used to simulate $[q, i_1, i_2]$ and $[q; p_1\dots p_{2n}; i_1, i_2]$, $i_1 + i_2 \ge 0$, from the rule set of M, respectively, while the rules of the form $[q; p_1\dots p_{2n}; 0, 0]$ are simulated by the rules $(b_1^{p_1+n}\dots b_n^{p_{2n}}, out; b_1^{-p_1}\dots b_n^{-p_n}, in)|_{\neg q'}$ in the second region. The maximal parallel application of these rules corresponds to the repeated application of the corresponding instructions of M. If a situation corresponding to a final state of M is reached by Γ, then it is possible to finish the computation by sending the symbol B from the second region to the first, which then results in exporting everything but C to the environment, reaching the final state $F_1 = BC$.

Let us now show that $\mathcal{L}(PFA) \subseteq \mathcal{L}(RRA)$. Let $\Sigma = \{a_i \mid i \ge 1\}$ be a countably infinite alphabet, and let $L \subseteq \Sigma^*$ be a language accepted by the P finite automaton $\Gamma = (V \cup \{a\}, \mu, (w_1, P_1, F_1), \dots, (w_m, P_m, F_m))$. We construct an RRA M, such that $L(M) = L(\Gamma)$.

Let $W = \bigcup_{1\le i\le m} w_i$ be the multiset of all objects from V which can be found in the membrane system initially, and let $c = max_{b\in V}(W(b))$. Let also $M = (\Sigma, Q, P, q_0, r_0, F)$ with $Q = \{0, 1, \dots c\}^n \times \{0, -1, \dots -c\}^n \times \{0, 1, \dots c\}^n$, and if we assume, without the loss of generality, that $V = \{b_1, \dots b_k\}$, then $n = m \cdot k$. The states of the finite control correspond to the possible distributions of elements from V in the different membranes of Γ, the initial state is $q_0 = (c_{1,1}, \dots, c_{1,k}, \dots, c_{m,1}, \dots, c_{m,k}, 0, \dots, 0)$ with $c_{i,j} = w_i(a_j)$, the set of final states is $F = \{q_f = (c_{1,1}, \dots, c_{m,k}, 0, \dots, 0) \in Q \mid$ for all i, $1 \le i \le m$, with $F_i \ne \emptyset$, $c_{i,j} = u_i(a_j)$, $1 \le j \le k$, for some $u_i \in F_i\}$. We say that a promoter or inhibitor Z of a rule $(x, in; y, out)|_Z \in P_i$ is consistent with a state

$q = (c_{1,1}, \ldots, c_{1,k}, \ldots, c_{m,1}, \ldots, c_{m,k}, c_{m+1,1}, \ldots, c_{3m,k})$, if it is consistent with the configuration of Γ corresponding to the state q, that is, with the configuration $(\bigcup_{1 \le i \le k} b_i^{c_{1,i}}, \ldots, \bigcup_{1 \le i \le k} b_i^{c_{m,i}})$ given by the first n coordinates of q.

The initial register contents is $r_0 = w_1(a)$, and the set of rules P is defined as follows.

$$
\begin{aligned}
P = \{&[q; i_1, i_2] \mid (a^{i_1}, out; a^{i_2}, in)|_Z \in P_1,\ q \in Q \text{ such that } Z \text{ is consistent} \\
&\text{with } q\} \cup \\
&\{[q; p_{1,1} \ldots p_{2m,k}; 0, 0] \mid p_{i,j} = -u(b_j),\ p_{parent(i)+n,j} = u(b_j), \\
&p_{i+n,j} = v(b_j),\ p_{parent(i),j} = -v(b_j) \text{ for all } (u, out; v, in)|_Z \in P_i, \\
&\text{such that } Z \text{ is consistent with } q,\ 1 \le i \le m,\ 1 \le j \le k\} \cup \\
&\{[q; p_{1,1} \ldots p_{2m,k}; i_1, i_2) \mid p_{1,j} = -u(b_j),\ (a^{i_1}u, out; a^{i_2}, in)|_Z \in P_1, \\
&\text{with } i_1 + i_2 > 0, u(a) = 0, u \ne \varepsilon,\ 1 \le j \le k, \text{ and } p_{i,j} = 0 \\
&\text{for } 2 \le i \le m,\ 1 \le j \le k, \text{ suchthat } Z \text{ is consistentwith } q\}.
\end{aligned}
$$

To see how these rules simulate the P finite automaton Γ, consider the following. A configuration $(a^{r_1}v_1, \ldots, v_m)$, $v_i \in V^\circ$, $1 \le i \le m$, of Γ corresponds to the configuration $(q, r_1, 0)$ of M where q describes the distribution of elements of V in the membrane structure, $q = (c_{1,1} \ldots, c_{m,k}, 0 \ldots, 0)$ with $c_{i,j}$ denoting the number of elements of symbol b_j inside membrane i, that is, with $c_{i,j} = v_i(b_j)$, $1 \le i \le m$, $1 \le j \le k$. Each rule of the P finite automaton corresponds to an instruction of M, their maximal parallel application is simulated by the repeated application of these instructions, and the final states of M are constructed to describe the final configurations. □

4 $\mathcal{L}(PFA)$ as the Extension of the Regular Language Class to Infinite Alphabets

First we show that all "conventional", finite alphabet regular languages can be accepted by P finite automata.

Lemma 2. $\mathcal{L}(REG) \subset \mathcal{L}(PFA)$.

Proof. Let $M = (\Sigma_1, Q, \delta, q_0, F)$ be a finite automaton over the finite input alphabet $\Sigma_1 = \{a_1, \ldots, a_k\}$, with set of states Q, transition relation $\delta : Q \times \Sigma \to Q$, initial state $q_0 \in Q$, and set of final states $F \subseteq Q$. Let the regular language accepted by M be denoted by $L(M)$. Let us also assume that $q_0 \notin F$.

Let $TRANS = \{[q_1, a_i, q_2] \mid \delta(q_1, a_i) = q_2\}$ and let $TRANS' = \{[q_1, a_i, q_2]' \mid \delta(q_1, a_i) = q_2\}$, a primed and a non-primed set of triples corresponding to the transitions of M. Let us also denote for any $t' \in TRANS'$, by $next(t') \in TRANS$ the set of those non-primed transition symbols which correspond to transitions that can follow the transition denoted by the primed symbol t', that is, $next(t') = \{[q_2, a_j, q_3] \in TRANS \mid t' = [q_1, a_i, q_2]'\}$. For any non-primed $t \in TRANS$, we always denote the corresponding primed symbol from $TRANS'$ by t'.

Now we construct a P finite automaton Γ, such that $L(\Gamma) = L(M)$. Let $\Gamma = (V \cup \{a\}, [\ ]_2\ ]_1, (w_1, P_1, \emptyset), (w_2, P_2, F_2))$ where $V = TRANS \cup TRANS' \cup \{\#\}$ and

$$
\begin{aligned}
w_1 &= a\#,\\
P_1 &= \{(a^i, in; a, out)|_t, (a^{i-1}, out)|_{t'} \mid t = [q_j, a_i, q_k],\ i > 1\} \cup\\
&\quad \{(a, in; a, out)|_t \mid t = [q_j, a_1, q_k]\},\\
\\
w_2 &= \{\{t, t' \mid t \in TRANS\}\},\\
P_2 &= \{(\#, in; t_0, out) \mid t_0 = [q_0, a_i, q]\} \cup\\
&\quad \{(t, in; t', out), (t', in; s, out) \mid t \in TRANS,\ s \in next(t')\},\\
F_2 &= \{\ \{\{t, t' \mid t \in TRANS\}\} - \{\{s'\}\} \mid \text{for}\\
&\qquad \text{all } s' \in TRANS' \text{ such that } s' = [q, a_i, q_f]',\ q_f \in F\}.
\end{aligned}
$$

It is not difficult to see how Γ simulates M. The rules of the second region are responsible for sending symbols representing the transitions of M into the first region in an order which is a legal transition sequence of M, and the rules of the first region import from the environment the necessary number of as corresponding to the input symbol belonging to the simulated transition. □

Now we show that the class of finite alphabet languages accepted by P finite automata is precisely the class of regular languages.

Lemma 3. *If L is a language over a finite alphabet, such that $L \in \mathcal{L}(PFA)$, then $L \in \mathcal{L}(REG)$.*

Proof. Let $M = (\Sigma, Q, P, q_0, r_0, F)$ be a RRA and let $L(M) \subseteq \Sigma_1^*$ where $\Sigma_1 \subseteq \Sigma$ is a finite alphabet. Note that the existence (the applicability) of a rule of the form $[q, 0, i_2] \in P$ with $i_2 > 0$ would contradict the fact that $L(M)$ is a language over a finite alphabet, thus, we can assume that there are no rules of this form in the rule set P. Let us also assume, without the loss of generality, that $q_0 \notin F$. In the following, we construct a nondeterministic finite automaton M', such that $L(M') = L(M)$.

Let for all $q \in Q$, $Dec_q = \{i_1 \mid [q'; i_1, i_2] \in P, q' \sim q, i_1 > 0\}$, and let ∞ be a symbol, such that for all $i \in \mathbb{N}$, $i \leq \infty$. Then, let $rest(q) \in \mathbb{N} \cup \{\infty\}$ be the set

$$
rest(q) \begin{cases} min(Dec_q) - 1 & \text{if } Dec_q \neq \emptyset,\\ \infty & \text{if } Dec_q = \emptyset, \end{cases}
$$

that is, $rest(q)$ denotes the maximal nonnegative integer value which can be stored in the first register when M is in state q, such that no rule of the form $[q', i_1, i_2]$ decrementing the first register can be applied because the contents of the first register is less then necessary. If for a certain $q \in Q$, there is no such

rule, that is, the set Dec_q is empty, then $rest(q) = \infty$. Note that for $q, q' \in Q$, if $q \sim q'$, then $rest(q) = rest(q')$.

Let $M' = (\Sigma_1, Q', \delta, q'_0, F')$ be a finite automaton with input alphabet Σ_1, state set Q', transition relation $\delta : Q' \times \Sigma_1 \to 2^{Q'}$, initial state $q'_0 \in Q'$, and set of final states $F' \subseteq Q'$.

The states of M' are elements of $Q \times \mathbb{N}$,

$$Q' = \{(q,i) \mid q \in Q, 0 \leq i \leq \max(\{rest(q) \mid q \in Q,\ rest(q) \neq \infty\}) + |\Sigma_1| + r_0 + c \cdot m \cdot |P|\},$$

where $c \in \mathbb{N}$, such that $Q \subseteq \{0,1,\ldots,c\}^n \times \{0,-1,\ldots,-c\}^n \times \{0,1,\ldots,c\}^n$ and $m = \max(\{i_2 \mid [q; p_1 \ldots p_{2n}; i_1, i_2] \in P\})$.

The initial state of M' is $q'_0 = (q_0, r_0)$. For the initial state, we define for all $j,\ 0 \leq j \leq |\Sigma_1|$, that is, since in our notation $a_0 = \varepsilon$, for all symbols from $\Sigma_1 \cup \{\varepsilon\}$

$$trans((q_0, r_0), a_j) = \{(q,i) \mid (q_0, r_0, 0) \vdash^{a_j} (q, i, 0)\}.$$

Let $Q'_0 = \{(q_0, r_0)\}$, and let $Q_1 = Q'_0 \cup \{(q,i) \in trans((q_0, r_0), a_j) \mid 0 \leq j \leq |\Sigma_1|\}$. Now, for all states in $(q,i) \in Q'_1$ and $0 \leq j \leq |\Sigma_1|$,

$$trans((q,i), a_j) = \{(q', i') \mid (q, i, 0) \vdash^{a_j} (q', i', 0)\},$$

and $Q'_2 = \bigcup_{(q,i) \in Q'_1, 0 \leq j \leq |\Sigma_1|} trans((q,i), a_j)$.

We continue this way the definition of $trans$ and $Q'_i,\ i \geq 0$. That is, if we already have Q'_i, then $Q'_{i+1} = \bigcup_{(q,k) \in Q'_i, 0 \leq j \leq |\Sigma_1|} trans((q,k), a_j)$, where for $0 \leq j \leq |\Sigma_1|$,

$$trans((q,k), a_j) = \{(q', k') \mid (q, k, 0) \vdash^{a_j} (q', k', 0)\}.$$

We claim that $\bigcup_{i \geq 0} Q'_i$ is a finite set, namely that $\bigcup_{i \geq 0} Q'_i \subseteq Q'$, that is, that M' with $\delta = trans$ is a finite automaton.

To see this, consider a derivation of the RRA M,

$$(q_0, r_0, 0) \vdash^{x_1} \ldots (q_i, r_i, 0) \vdash^{x_{i+1}} (q_{i+1}, r_{i+1}, 0) \ldots \vdash^{x_n} (q_n, r_n, 0),$$

and let $(q_i, r_i, 0) \vdash^{x_{i+1}} (q_{i+1}, r_{i+1}, 0)$ be a transition with

$$(q_i, r_i, 0) \Rightarrow^* (q_i, r'_i, j) \Rightarrow^{x_{i+1}} (q_{i+1}, r_{i+1}, 0) \tag{2}$$

where $x_{i+1} = a_j$, and $r_{i+1} = r'_i + j$.

There are two possibilities.

1. If $rest(q_i) < r_i$, then $r'_i \leq rest(q_i)$, otherwise there would be a rule $[q', k_1, k_2] \in P$ with $q' \sim q,\ k_1 > 0$, such that $r'_i \geq k_1$, and this is not allowed by the definition of a derivation step of RRA. So, since $r'_i \leq rest(q_i)$, the new contents of the first register is $r_{i+1} = r'_i + j \leq rest(q_i) + j \leq rest(q_i) + |\Sigma_1|$.
2. If $rest(q_i) > r_i$ then $r'_i = r_i - s_{i,1},\ j = s_{i,2}$ where if P_i is the multiset of rules of the form $[q'_i, p_1 \ldots p_{2n}; i_1, i_2]$ applied in the transition denoted by (2) above,

then $s_{i,1} = \sum_{[q;p_1 \dots p_{2n};i_1 i_2] \in P_i} i_1$, $s_{i,2} = \sum_{[q;p_1 \dots p_{2n};i_1 i_2] \in P_i} i_2$ (since there is no rule of the form $(q, 0, i) \in P$). Now, since $r_{i+1} = r_i - s_{i,1} + s_{i,2}$, the value of the first register can increase at most with the value $s_{i,2} \leq m_i \cdot |P_i|$ during the transition, where $m_i = \max(\{i_2 \mid [q; p_1 \dots p_{2n}; i_1, i_2] \in P_i\})$.

Let us now look again at the derivation and the transition step $(q_i, r_i, 0) \vdash^{x_{i+1}} (q_{i+1}, r_{i+1}, 0)$ above. If this transition is of the first type, then $r_{i+1} \leq rest(q_i) + |\Sigma_1|$ which means that $(q_{i+1}, r_{i+1}) \in Q'$.

If the transition step $(q_i, r_i, 0) \vdash^{x_{i+1}} (q_{i+1}, r_{i+1}, 0)$ is of the second type, then let $(q_j, r_j, 0) \vdash^{x_{j+1}} (q_{j+1}, r_{j+1}, 0)$ for some $l \leq j \leq i$ the be the transition sequence containing all consecutive transitions of the second type which took place before, that is, where

- either $l = 0$, or $(q_{l-1}, r_{l-1}, 0) \vdash^{x_l} (q_l, r_l, 0)$ is a transition of the first type, and
- $(q_j, r_j, 0) \vdash^{x_{j+1}} (q_{j+1}, r_{j+1}, 0)$ is a transition of the second type for all $l \leq j \leq i$.

Since the rules $[q'_i; p_1 \dots p_{2n}; i_1, i_2)$, $q'_i \sim q_i$, with $i_2 > 0$ have the property that no $q''_i \sim q'_i$, $q''_i \in \{0, 1, \dots, c\}^n \times \{0\}^{2n}$ is reachable after their application, if we start the derivation with a register value r_l in the first register, then the total increase of this value cannot be more then $s_2 = c \cdot m \cdot |P|$ where $m = \max(\{i_2 \mid [q; p_1 \dots p_n; i_1, i_2] \in P\})$. Thus $r_{i+1} \leq r_l + s_2$. Since $r_l = r_0$, or $r_l \leq rest(q_i) + |\Sigma_1|$, it has to hold that $r_{i+1} \leq r_0 + rest(q_i) + |\Sigma_1| + s_2$,and therefore $(q_{i+1}, r_{i+1}) \in Q'$ holds also in this case.

After these considerations we can see that *trans* is a function of type $Q' \times \Sigma_1 \to 2^{Q'}$, thus, if we take $\delta = trans$ and $F' = \{(q, i) \mid (q, i) \in Q',\ q \in F\}$, then it is clear that the finite automaton M' simulates the RRA M, because δ specifies the transitions of M and all states of F' correspond to accepting configurations of M. □

5 Conclusion

We have presented two equivalent computing models which are able to characterize languages over infinite alphabets, we called them P finite automata and restricted register finite automata. We have shown that the languages over finite alphabets contained in the language class that these models characterize are precisely the regular languages, thus it can be seen as the extension of the class of regular languages to infinite alphabets. Without going into the details, we would like to add that the languages mentioned in the introduction as regular in the sense of [3] but not regular in the sense of [5], and vice versa, can all be accepted by our model, thus, it seems that our approach is able to eliminate at least some of the shortcomings of previous attempts to define the class of regular languages over infinite alphabets in a reasonable way. A more detailed analysis of $\mathcal{L}(PFA)$ (or equivalently $\mathcal{L}(RRA)$) remains a topic of further study.

References

1. Cheng, E.H.Y., Kaminski, M.: Context-free Languages over Infinite Alphabets. *Acta Informatica*, **35** (1998), 245–267.
2. Csuhaj-Varjú, E., Vaszil, Gy.: P Automata. In: Păun, Gh., Zandron, C. (eds.): *Pre-Proceedings of the Workshop on Membrane Computing WMC-CdeA 2002*, Curtea de Argeş, Romania, August 19-23, 2002. Pub. No. 1 of MolCoNet-IST-2001-32008 (2002) 177–192, and also in Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C. (eds.): *Membrane Computing*. LNCS 2597, Springer, Berlin, 2003, 219–233.
3. Kaminski, M., Francez, N.: Finite-Memory Automata. *Theoretical Computer Science*, **134** (1994), 329–363.
4. Martín-Vide, C., Păun, A., Păun, Gh.: On the Power of P Systems with Symport Rules. *Journal of Universal Computer Science*, **8**(2) (2002), 317–331.
5. Otto, F.: Classes of Regular and Context-free Languages over Countably Infinite Alphabets. *Discrete Applied Mathematics*, **12** (1985), 41–56.
6. Păun, A., Păun, Gh.: The Power of Communication: P Systems with Symport/Antiport. *New Generation Computing*, **20**(3) (2002), 295–306.
7. Păun, Gh.: *Membrane Computing: An Introduction*. Springer, Berlin, 2002.
8. Rozenberg, G., Salomaa, A. (eds.): *Handbook of Formal Languages*. Springer-Verlag, Berlin, vol. 1-3, 1997.

Mitotic Oscillators as MP Graphs

Giuditta Franco[1], Pietro Hiram Guzzi[2],
Vincenzo Manca[1], and Tommaso Mazza[2]

[1] Department of Computer Science, University of Verona, Italy
franco@sci.univr.it, vincenzo.manca@univr.it
[2] Magna Græcia University of Catanzaro, Italy
{hguzzi, t.mazza}@unicz.it

Abstract. This paper proposes a model in terms of metabolic P graphs of a few important processes occurring during the biological phase where the choice is made to begin again mitosis or to arrest it. The cellular processes during this phase turn out to be especially interesting in the case of DNA damage, which triggers a specific destruction of Cdc25A phosphatase. It has important implications to understand the role of cell cycle checkpoints and the mechanism(s) guiding the proliferation of UV-resistant tumored cells. The formalism of metabolic P graphs highlights the relevant information of the biological network dynamics, and the individuation of few parameters rules the basic mechanisms of Cdc25A degradation, involving a couple of important mitotic oscillators.

1 Introduction

Membrane systems were proposed as a model to represent various aspects of molecular localization and compartmentalization, including the movement of molecules between compartments, the dynamic rearrangement of molecular reactions, and the interaction between molecules in a compartmentalized setting. They have been widely investigated as models and tools of interest for computer science [20], while recently the intent to employ membrane systems as a framework to study real biological systems is vividly pursued [4,1,3].

In particular, a novel perspective has been introduced by Manca and his group [4,18] along with the idea of controlling the evolution of a (membrane) system by means of rules whose "strength" is identified with a numerical value (reactivity) which depends on the system state (that is, the objects concentration). Every reactivity denotes the ability of the corresponding rule to compete against other rules in capturing part of the population of reactants on which the reaction is performed [7]. By going ahead in this perspective, every reactivity is determined by the corresponding (reaction) map evaluated on the state of the system, and a strategy for partitioning the objects in the system (at every transition) is given, which depends on the relative magnitude of every reactivity [5]. This new strategy of rule application was inspired by actual 'metabolic reactions', and it seems to lead membrane computing toward interesting simulations of biological

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 382–394, 2006.

processes, such as representations of signal transduction networks and complex oscillations [6,8,16].

The dynamics regulating the cyclic oscillation of some biochemicals is especially important to figure out the regulation mechanisms that provide the life of the cellular system. Namely mitotic oscillations are a mechanism exploited by nature to regulate the mitosis process, that is, the cell division aimed at producing two daughter cells identical to the single parent cell. Mitotic oscillations concern the fluctuation of activation state of the substances involved in the process. The context and the inspiration of this paper lie in [17], where the mitotic oscillator of amphibian embryos was studied by means of three models, inspired by the A. Goldbeter differential equations system and by the careful observation and the direct description of the biological oscillator itself. In particular, the formalism of metabolic P graphs is introduced that represents all the information needed by the metabolic algorithm to calculate the dynamics of a biological network.

Here we analyze the interruption, after a DNA damage, of the cell division cycle due to the degradation of Cdc25A, which is a phosphatase crucial in the mitosis process [14,15]. This is even more interesting if one considers that up to now the arrest induced by DNA damage has been ascribed only to the transcription factor and tumor suppressor protein p53. Surprisingly though, transient inhibition of Cdk2 (the kinase whose complex is activated by Cdc25A) in response to DNA damage occurs even in cells lacking p53 [11] or p21, [24] which is an inhibitory protein transcriptionally regulated by p53. Thus, $p21^{WAF1/CIP1}$ is an important effector of the mitosis arrest [9] and plays a critical role in the well-documented p53 function (for a P systems model of p53 signaling pathways by we refer to [22]). It has also important implications for understanding cell cycle checkpoints and the mechanism(s) through which p53 inhibits human neoplasia.

Given the importance of checkpoints for preventions of genetic diseases including cancer, we explore these alternative mechanisms of mitosis arrest, by identify a p53-independent signalling pathway that causes the cell division arrest after DNA damage. We formalize such a biological reality as a metabolic P graph [17], with the aim to obtain a model to reproduce and observe the evolution of the system. The fluctuation of the key elements concentrations is crucial to capture typical healthy states of the system. The goal is to deflect whichever diseased path to healthy paths by modulating, in a few steps, the considered concentrations.

In what follows, we first we present a qualitative description of the main phenomena involving three mitotic oscillators, then the pathway in which we are interested, and finally the graph describing the corresponding process.

2 Checkpoints in Cell Cycle

The cell cycle consists of the four phases, Gap 1, S, Gap 2, and M, that are displayed in Figure 1. Gap 1 (called **G1**) is the interval between mitosis and DNA replication, that is characterized by cell growth. G1 is a vital phase of cell growth because just in this cell cycle phase the choice to begin again mitosis is made.

The transition that occurs at the restriction point (called **R**) during the G1 phase commits the cell to the proliferative cycle. If the conditions that enforce this transition are not present, the cell exits the cell cycle and enters a non-proliferative phase (called **G0**) during which cell growth, segregation, and apoptosis occur [12,2]. Replication of DNA occurs during the synthesis phase (called **S**), which is followed by a second gap phase (called **G2**) during which growth and preparation for cell division occurs. Mitosis and production of two daughter cells occur in the phase called **M**. The switches from one phase to the next one are critical checkpoints of the basic cyclic mechanism of proliferating cells, and they are studied with a wide interest [11,13,24].

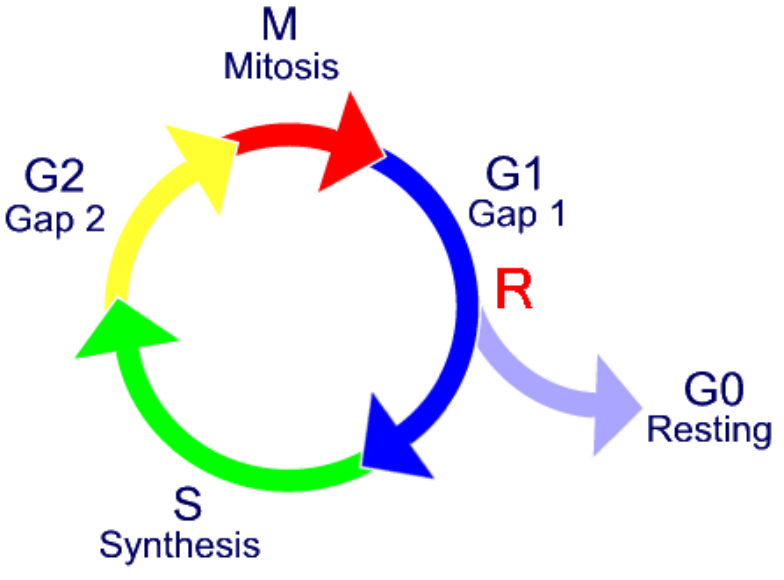


Fig. 1. Phases of the cell cycle. During the G1 phase progression or arrest (G0 phase) of the cycle is decided.

The passage through the four phases of the cell cycle is regulated by a family of *cyclins*[1] that act as regulatory subunits for *cyclin-dependent kinases cdks*.

Cyclins are a family of proteins involved in the progression of cells through the whole cell cycle. Cyclin forms a complex with the cyclin-dependent kinase, which activates the latter protein kinase function and promotes the mitosis phase. When cyclin concentration in the cell is low, it detaches from Cdk inhibiting the enzyme's activity (probably by producing a protein chain to block the enzymatic site). Therefore, the activity of the various *cyclin/cdk* complexes that regulate the progression through G1-S-G2 phases of the cell cycle is controlled by the

[1] Cyclins are so named because their concentration varies in a cyclical fashion during the cell cycle. They are produced or degraded as needed in order to drive the cell through the different phases of the cell cycle.

synthesis of the appropriate cyclins during a specific phase of the cell cycle. In [17] one may find a few models for the minimal structure of such a mitotic oscillator, that is, the binding of the cyclin with the cdk2 which so passes from its inactive to its active state.

The cyclin/cdk complex is then activated by the sequential phosphorylation and dephosphorylation of the key residues of the complex located principally on the cdk subunits. In eukaryotes, protein phosphorylation is probably the most important regulatory event. Many enzymes and receptors are switched "on" or "off" by phosphorylation and dephosphorylation. Phosphorylation is catalyzed by various specific protein kinases, whereas phosphatases dephosphorylate. In the framework of the process we are going to describe in the next section, we have three oscillators: a DNA-damage induced activation of the kinases Chk1 and Chk2, and a consequent activation of Cdc25A (a serine/threonine phosphatase) due to its phosphorylation (phosphoCdc25A) made by the active Chk1 and Chk2. Finally, a massive amount of phosphoCdc25A dephosphorylates the complex cdk2-cyclinE by making it active and inducing the S-phase of cell division (see Figure 2).

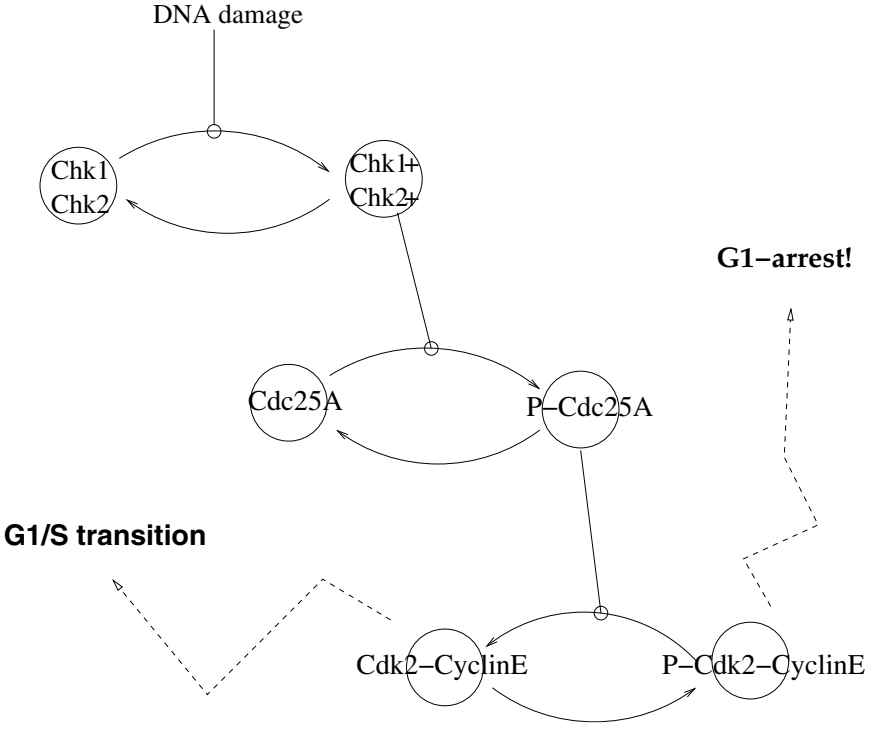


Fig. 2. Passages from the inactive states to the active ones in our process – oscillators are all reversible reactions

We will examine the role of Cdc25 family members of which at least Cdc25A is essential both for the entry into S phase [14], at the checkpoint control of the G1-S transition, and for the cell cycle arrest in response to a DNA damage. In particular, we analyze the degradation of phosphorylated Cdc25A by ubiquitin mediation, which inhibits the activation of the complex cdk2-cyclinE and provokes the G1 arrest. Such a degradation takes place in the *cytosol* (which is the fluid part of the cytoplasm, different from organelles and membranes) and is mediated by the 'endopeptidase activity' of '26S proteasome', causing the dissociation of phosphoCdc25A in its constituting aminoacids (Figure 5). This mechanism is intriguing because we could learn how to module the quantity of Cdc25A in order to arrest the proliferation of tumored cells (that have DNA damage).

3 Cycle Arrest in Response to Stress

Episodes of DNA damage during G1 pose a particular challenge, because replication of damaged DNA can be deleterious and because no other chromatid is present to provide a template for recombinational repair. Besides, by considering that cyclins operate as a promoting factor for the mitosis phase and that typical cancer evolutions act as a suppressor of certain components of the cyclins family, in case of DNA damage the desired (healthy) state is identified by the G0 phase. Thus, in this context, we are interested to figure out the conditions under which G0 is reached.

There are several proteins that can inhibit the cell cycle in G1 but, when DNA damage has occurred, p53 is that protein which accumulates in the cell inducing the p21-mediated inhibition of cyclin D/cdk. In fact, DNA-damaging agents induce a p53-dependent G1 arrest that may be critical for p53-mediated tumor suppression [23]. There is an alternative way though, where inhibition of Cdk2 in response to DNA damage occurs even in cells lacking p53 or p21; this is depicted in Figure 3.

Human cells respond to ultraviolet light or ionizing radiation by rapid, ubiquitin and proteosome-dependent protein degradation of Cdc25A[15,13]. Namely, DNA damage triggers specific destruction of Cdc25A phosphatase. This event prevents the entry of a cell into the S-phase, by maintaining the cyclin E-Cdk2 complexes in phosphorylated form [19]. "Unfortunately", between 16 and 24 hours after the exposure to UV, the cells resume DNA replication and progression through the cell cycle, indicating that the UV-induced cell cycle arrest is reversible.

The overall phenomenon described here is not dependent on p53 and previous studies [12] have demonstrated that the abundance of Cdc25A appears to determine the extent of DNA synthesis upon UV-induced DNA damage. Thus, the elimination of Cdc25A evokes a cell cycle arrest promoting repair of the DNA crosslinks caused by UV and protects the cells from formation of the DNA strand breaks. Finally, a 3-hour period of expression of Cdc25A to prevent downregulation of the cellular Cdc25A activity by UV reduced the survival of the

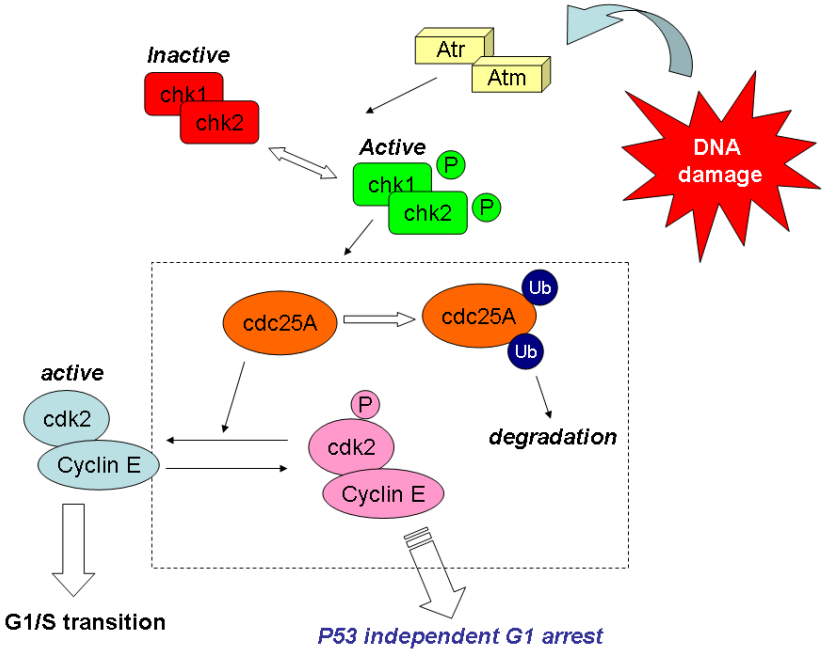


Fig. 3. p53 independent G1 arrest in response to stress

irradiated cells examined by colony formation assays. These results uncover a mechanism of cellular defense against genotoxic stress that can be rewritten in mathematical way and subsequently simulated.

The substances relevant to describe the process of degradation of Cdc25A in the cytosol, after its activation in the nucleus of the cell, are reported in Table 1.

The relevant chemical reactions of the process (that represent the stoichiometric level of a metabolic P graph) are the following. First the phosphorylation of Cdc25A at ser123 in response to DNA damage occurs in the nucleoplasm.

Nucleoplasm

$$ATP + Cdc25A + Chk1 \rightarrow ADP + phosphoCdc25A \qquad (J_0)$$

$$ATP + Cdc25A + Chk2 \rightarrow ADP + phosphoCdc25A \qquad (J_1)$$

The rules J_0 and J_1 activate the phosphatase Cdc25A by means of active kinases. In such a way, the detection of DNA damage results in the phosphorylation of Cdc25A at Ser-123 by Chk1 and Chk2, by concurrently inhibiting Cdc25A. Then, phosphoCdc25A migrates from the nucleoplasm to the cytosol of the cell; for notational convenience we call it phosphoCdc25A@Cytosol, and we have the following rule.

Table 1. Actors of the model

Symbol	Name	Compartment
Chk1	Checkpoint kinase 1	nucleoplasm
Chk2	Serine/threonine-protein kinase Chk2	nucleoplasm
ATP	Adenosine Triphosphate	nucleoplasm
Cdc25A	M-phase inducer phosphatase 1	nucleoplasm
ADP	Adenosine diphosphate	nucleoplasm
phosphoCdc25A	M-phase inducer phosphatase 1	nucleoplasm
Ubiquitin	Ubiquitin	cytosol
UbiqutinLigase	E3 Ubiquitin Ligase	cytosol
Ubiquitinatedphospho--25Cdc25A	M-phase inducer phosphatase 1	cytosol
proteasome26s	Proteosome complex	cytosol

Migration to Cytosol

$$phosphoCdc25A \rightarrow phosphoCdc25A@Cytosol \qquad (J_2)$$

Finally, in the cytosol we have sequentially the degradation of phosphorylated Cdc25A by ubiquitin mediation and the degradation of Cdc25A by proteasome, that may be described by the following J_3 and J_4 rules, respectively.

Cytosol

$$phosphoCdc25A@Cytosol + Ubiquitin + UbiquitinLigase \rightarrow \\ UbiquitinatedphosphoCdc25A@Cytosol + UbiquitinLigase \qquad (J_3)$$

$$UbiquitinatedphosphoCdc25A@Cytosol + proteasome26s \rightarrow \\ AminoAcid + proteasome26s. \qquad (J_4)$$

The J_3 reaction is mediated by the 'ubiquitin-protein ligase activity' of the catalyzing enzyme 'Ubiquitin ligase', and one molecule of 'Ubiquitinated PhosphoCdc25A' is produced by transforming one molecule of phosphoCdc25A@Cytosol. The reaction J_4 finally degrades UbiquitinatedphosphoCdc25A@Cytosol by transforming it into one molecule of its aminoacid by means of proteasome 26s activity.

The model of the above interactions may be depicted as in Figure 4 where reactants reside in compartments (nucleoplasm and cytosol) and interact between them (arrows). However, no indication is given about the dynamics of the system, that is *how* the reactants interact to each other. In order to point out the dynamics of the system, in the next section we describe the process of interest by means of a metabolic P graph, a formalism introduced in [17] which extends the *Stoichiometric Network Analysis* [10] developed in the context of complex reaction networks [21].

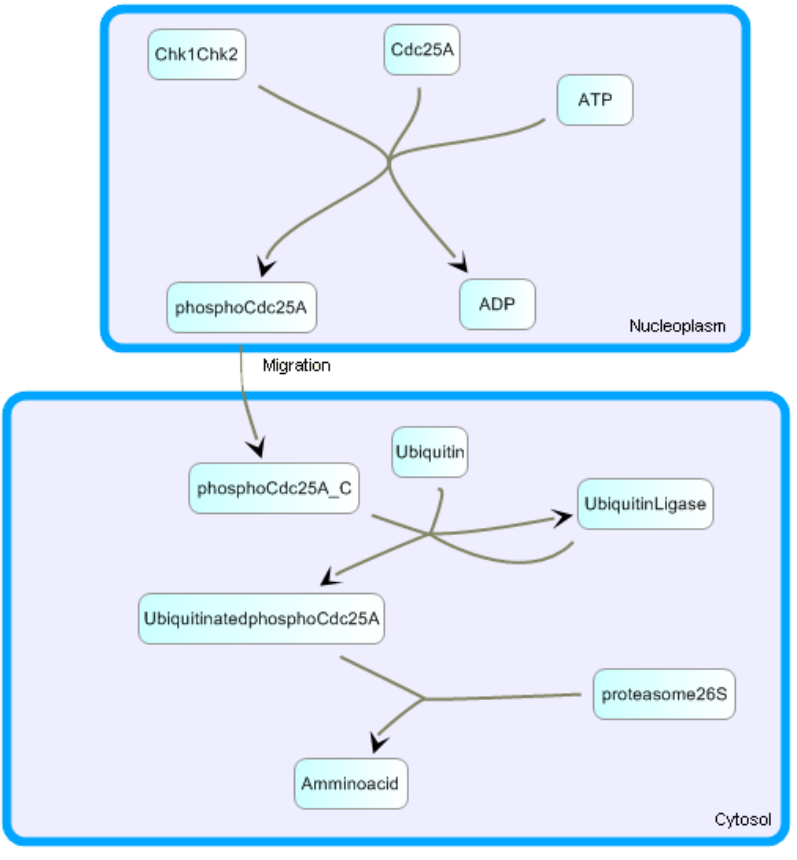


Fig. 4. Graph of interactions

4 Metabolic P Graphs

The starting assumption is that the localization and the concentrations of any biochemical element at each instant determines all the relevant properties which underly the function that a biological system exhibits at that (observation) time.

By definition [17], an *MP graph* is a structure $G = (T, R, F, E, C)$, where:

- T is the set of nodes representing substances relevant for the process (for example, those in Table 1). We may think of each element in T as the container of a certain amount of a peculiar kind of substance. We represent such a kind of nodes as circles with the type of objects contained in it (see Figure 5).
- R is the set of nodes representing biochemical reactions between substances. We represent each of the nodes in R as a full bullet and we label it with the name of the reaction represented by that node (see Figure 5).
- F is the set of nodes labeled by reaction maps represented in Figure 5 along with dotted arrows.

- E is a set of nodes presenting input or output gates. It usually contains two different kind of nodes: input gates and output gates. In our case, some non-definite substances induced by DNA damage are assumed to come through an input gate (denoted by a star in our graph), and the output gates are represented by relevant effects of the process, such as mitosis, arrest, and degradation (see Figure 5).
- C is a set of edges (connections) between nodes. The edges are of two different kinds: plain edges or dashed edges.
  i) Plain edges connect types to biochemical reactions (circles and full bullets in the graph), in particular, they specify *reactants* and *products* of the reactions. Arcs connecting reactants to reactions are depicted as lines while arcs connecting reactions to products appear as arrows (oriented arcs).
  ii) Dotted arrows connect a possibly empty set of substances to one full bullet (they represent the reaction map of the corresponding rule).

Two components are easily distinguishable in MP graphs: a *stoichiometric* component and a *regulation* component. The stoichiometric component is the subgraph obtained after removing from an MP graph $G = (T, R, F, E, C)$ the nodes F and the dotted arcs which connect them to the other nodes. This removed part is the *reaction regulation layout* of G.

In Figure 5 we abstract from the graph in Figure 4 the stoichiometric component of the system, and we insert the regulation component as a way to control the process. Note that in this model the rule J2 of the previous section is not present because it is not relevant for the dynamics of the substances variations, while J1 and J2 are assembled in the rule R2 because they act in parallel and, by experimental observations, their reactivities may be considered identical.

As one can easily see, the formalism of MP graphs highlights the information of the biological network dynamics by pointing out the crucial tuning points of the system. In Figure 5, for example, there is an important difference with respect to the graph in Figure 4: the dynamics of the process is expressed by the graph and few essential parameters regulating the pathways are identified in the reactivities of the rules. Namely, the reactivities f1, f2, f3, and f4 control the dynamics of the activation oscillators of the kinases and the complex CyclinE-cdk2, respectively, while the reactivities h2, h3, and h4 control the information flow of PhosphoCdc25A degradation process. Unfortunately, the reactivities are the unknown part of the system, and there is work in progress to identify them and to simulate the dynamics of the graph by following the strategy of the metabolic P algorithm [17]:

- Reactants are distributed among all the rules step by step according to a "competition" strategy.
- If a couple of rules need the same reactant in two cases (R2 and K2, R3 and J3), then each of these rules gets a portion of the available substance, in a percentage that is proportional to its reaction strength (*reactivity*) at that step (f2 and h2, f3 and h3).
- The reactivity of a rule at a given instant depends on the state of the system, defined as the concentration and localization of all substances.

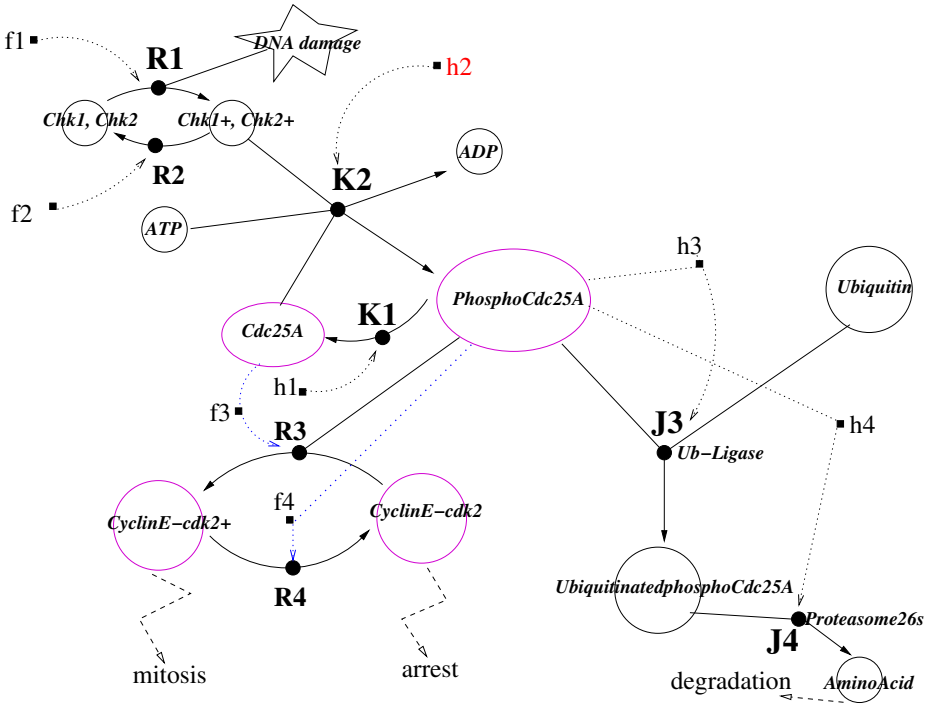


Fig. 5. Metabolic P graph of the process leading to a p53 independent G1 arrest. Rules K2, J3, J4, with their reactivities h2, h3, h4, are crucial for the Phospho-Cdc25A degradation process induced by UV radiation by means of DNA damage. The role of Ubiquitin Ligase and Proteasome26s that are catalyzing enzymes are identified with the reactions themselves (given the assumption that they are present in the cytosol in abundance for the process). Rules R1, R2, R3, R4 control the "back and forth" of the oscillators.

– According to its stoichiometric "reading", any rule determines its own reaction unit and therefore the amount of substances which it consumes and produces.

Our qualitative reasoning about the reactivities follows. The oscillator ($R1$, $R2$) in a first approximation may be neglected: the reactions velocities are such that we can suppose to have enough amount of activated $Chk1$ and $Chk2$ kinases to trigger the process. $K2$ is the highest reactivity in the whole system (for instance, $h2 = 1$), while $K1$ is very slow (for instance, $h1 = 0.1$). We know that the amount of (inactive) $Cdc25A$ is from 0 to 40 reaction units, and that about one third of it is degraded. More precisely, the process performed by J3 and J4 has reactivity 0.3 times the amount of cdc25A, that is, $h3 = h4 = 0.3|cdc25A|$. Finally, the oscillator ($R3, R4$) has the most complex reactivities: $f3 = \alpha|Cdc25A|$ and $f4 = \beta|PhosphoCdc25A|$, with α switching from assuming values in [0, 0.15] to assuming values in [0, 1], and β switching from assuming

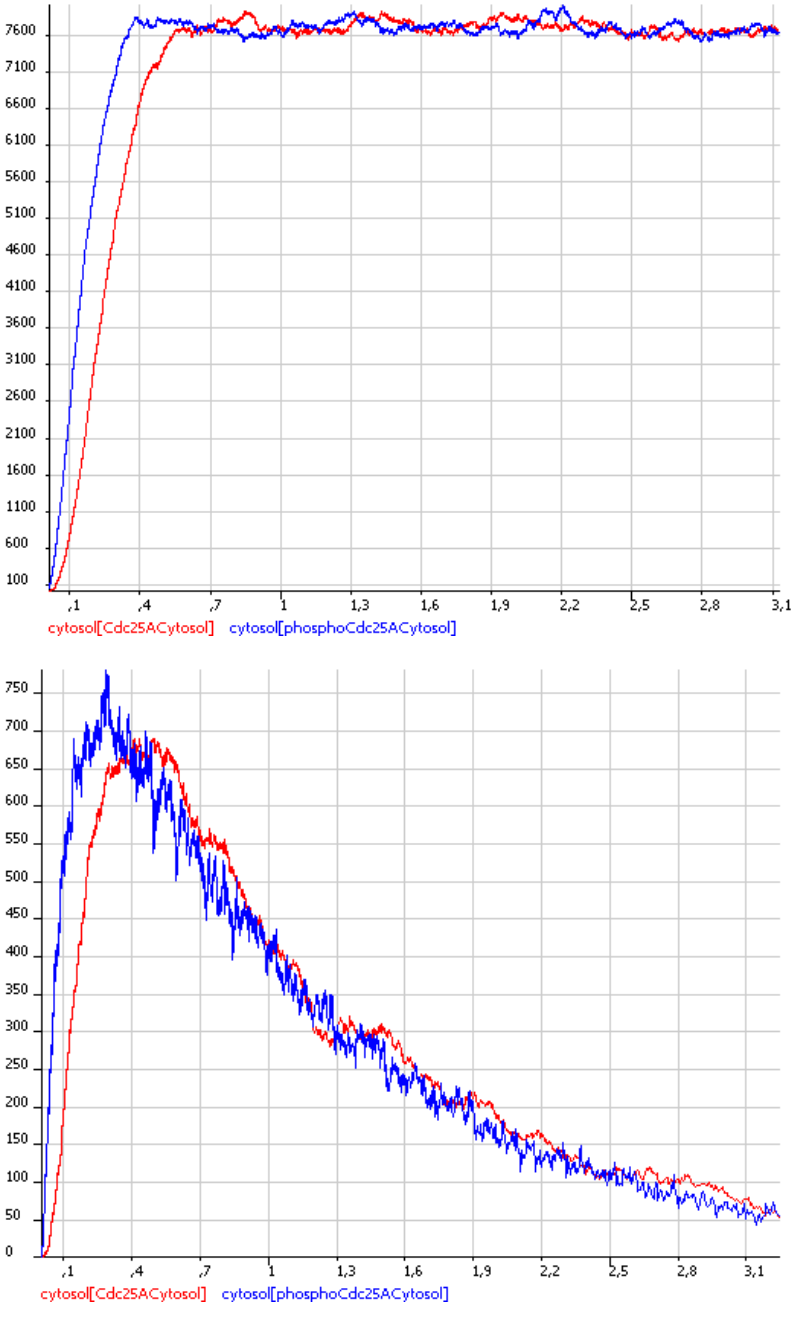


Fig. 6. Expected behavior of the (R3,R4) oscillator with low degradation (i.e., low amount of ubiquitin) and high degradation (high amount of ubiquitin), graph on the top and on the bottom respectively

values in [0, 1] to assuming values in [0, 0.25]. The desired goal of the simulations is to show that both the amounts of cdc25A and of the complex CyclinE-cdk2 oscillate, and these oscillations depend on the reactivity of $K2$, which seems to be the crucial value for this system dynamics.

The most important substances to consider are Cdc25A and PhosphoCdc25A, which are, respectively, responsible to arrest or to trigger the mitosis of the cell, by means of the activation of the complex CyclinE-cdk2. Their behavior is reported in Figure 6 when degradation in the cytosol is low (upper figure) and when degradation in the cytosol is high (bottom figure). In the first case we have a permanent oscillator, while in the second one both substances degrade if not fed by the environment. The conclusion is that the dynamics of the system and even the choice between mitosis or arrest is strongly influenced by the amount of ubiquitin. Therefore, an external intervention to increase such a substance would trigger an arrest of the mitosis cycle.

5 Future Work

We are further investigating two strategies to systematically compute the reactivities of MP metabolic graphs describing biological systems, namely using genetic algorithms, which find "good" values in order to obtain a desired behavior, and using the method of *MP Log-gain Regulation*, based on the computational progression of the MP algorithm starting by actual biological data.

The ultimate idea of this work is to insert this graph in a more general one which models also the pathways of p53 dependent G1 arrest, in such a way to figure out the relationships between the p53-mediated tumor suppression and the radiation-resistant DNA synthesis phase (and consequent cell mitosis).

References

1. N. Barbacari, A. Profir, and C. Zelinschi. Gene regulatory network modelling by means of membrane systems. In R. Freund, G. Lojka, M. Oswald, and Gh. Păun, editors, *Pre-proceedings of the 6th International Workshop on Membrane Computing, July 18-21, 2005, Vienna, Austria*, pages 162–178, 2005.
2. J. Bartek and J. Lukas. Pathways governing g1/s transition and their response to DNA damage. *FEBS Lett.*, 3(490):117–122, 2001.
3. F. Bernardini, M. Gheorghe, N. Krasnogor, R.C. Muniyandi, M.J. Pérez-Jiménez, and F.J. Romero-Campero. On P systems as a modelling tool for biological systems. In R. Freund, Gh. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, International Workshop, WMC6, Vienna, Austria, 2005, Selected and Invited Papers*, volume 3850 of *Lecture Notes in Computer Science*, pages 115–134. Springer, 2006.
4. L. Bianco, F. Fontana, G. Franco, and V. Manca. P systems for biological dynamics. In G. Ciobanu, Gh. Păun, and M.J. Perez-Jimenez, editors, *Applications of Membrane Computing*, chapter 3, pages 81–126. Springer, 2006.
5. L. Bianco, F. Fontana, and V. Manca. Metabolic algorithm with time-varying reaction maps. In *Proceedings of the Third Brainstorming Week on Membrane Computing, Sevilla, Spain*, pages 43–61, February 2005.

6. L. Bianco, F. Fontana, and V. Manca. P systems and the modeling of biochemical oscillations. In R. Freund, Gh. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 6th International Workshop, WMC 2005*, volume 3850 of *Lecture Notes in Computer Science*, pages 200–209. Springer, January 2006.
7. L. Bianco, F. Fontana, and V. Manca. P systems with reaction maps. *International Journal of Foundations of Computer Science*, to appear (2006).
8. L. Bianco and V. Manca. Symbolic generation and representation of complex oscillations. *International Journal of Computer Mathematics*, 17, 1 (2006), 27–48.
9. V. Chopin, R.A. Toillon, and N. Jouy. P21(waf1/cip1) is dispensable for g1 arrest, but indispensable for apoptosis induced by sodium butyrate in mcf-7 breast cancer cells. *Oncogene*, 23(1):21–29, 2004.
10. B. L. Clark. Stability of complex reaction networks. *Adv. Chem. Phys.*, 43:1–216, 1983.
11. J. D'Anna, J.G. Valdez, R.C. Habbersett, and H.A. Crissman. Association of g1/s-phase and late s-phase checkpoints with regulation of cyclin-dependent kinases in chinese hamster ovary cells. *Radiat. Res.*, 148:260–271, 1997.
12. K.I. Nakayama et al. Regulation of the cell cycle at the g1-s transition by proteolysis of cyclin e and p27kip1. *Biochem. Biophys. Res. Commun.*, 4(282):853–860, 2001.
13. J. Falck, N. Mailand, R.G. Syljuåsen, J. Bartek, and J. Lukas. The atm-chk2-cdc25a checkpoint pathway guards against radioresistant DNA synthesis. *Nature*, 410(6830):842–847, 2001.
14. S. Jinno, K. Suto, A. Nagata, M. Igarashi, Y. Kanaoka, H. Nojima, and H. Okayama. Cdc25a is a novel phosphatase functioning early in the cell cycle. *EMBO J.*, 13:1549–1556, 1994.
15. N. Mailand, J. Falck, C. Lukas, R.G. Syljuåsen, M. Welcker, J. Bartek, and J. Lukas. Rapid destruction of human cdc25a in response to DNA damage. *Science*, 288(5470):1425–1429, 2000.
16. V. Manca. Topics and problems in metabolic p systems. Fourth Brainstorming on Membrane Computing, Sevilla, Spain, 2006.
17. V. Manca and L. Bianco. Biological networks in metabolic p systems. Submitted.
18. V. Manca, L. Bianco, and F. Fontana. Evolutions and oscillations of P systems: Applications to biological phenomena. In G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 5th International Workshop, WMC 2004*, volume 3365 of *Lecture Notes in Computer Science*, pages 63–84. Springer, 2005.
19. E.A. Nilssen, M. Synnes, N. Kleckner, B. Grallert, and E. Boye. Intra-g1 arrest in response to uv irradiation in fission yeast. *Proc. Natl. Acad. Sci.*, 100(19):10758–63, 2003.
20. Gh. Păun. *Membrane Computing. An Introduction.* Springer, 2002.
21. L.A. Segel and I.R. Cohen. *Design Principles for the Immune System and Other Distributed Autonomous Systems.* Santa Fe Institute Studies in the Sciences of Complexity, Oxford University Press, 2001.
22. Y. Suzuki and H. Tanaka. Modelling p53 signaling pathways by using multiset processing. In G. Ciobanu, M. J. Pérez-Jiménez, and Gh. Păun, editors, *Applications of Membrane Computing*, Natural Computing Series, pages 203–214. Springer, Berlin, 2006.
23. T. Waldman, K.W. Kinzler, and B. Vogelstein. p21 is necessary for the p53-mediated g1 arrest in human cancer cells. *Cancer Research*, 55(22):5187–5190, 1995.
24. H. Zhao, J.L. Watkins, and H. Piwnica-Worms. Disruption of the checkpoint kinase 1/cell division cycle 25a pathway abrogates ionizing radiation-induced s and g2 checkpoints. *Proc. Natl. Acad. Sci.*, 99:14795–800, 2002.

Infinite Hierarchies of Conformon-P Systems

Pierluigi Frisco

School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh, EH14 4AS, UK
pier@macs.hw.ac.uk

Abstract. Two models of conformon-P systems, one restricted in the number of input conformons and the other restricted in the number of input membranes, are proved to induce infinite hierarchies.

The described systems do not work under the requirement of maximal parallelism and perform deterministic simulations of restricted counter machines.

1 Introduction

The subdivision of a cell into compartments delimited by membranes has been an inspiration to G. Păun for the definition of a new class of (distributed and parallel) models of computation called *membrane systems* [23].

The hierarchical structure, the locality of interactions, the inherent parallelism, and also the capacity (in the less basic models) for membrane division, represent the distinguishing hallmarks of membrane systems.

Research on membrane systems, also called 'P systems' (where 'P' stands for 'Păun'), has really flourished [24].

One can distinguish three main lines of research concerning membrane systems:

1. establishing their generative power;
2. using them to develop algorithms for solving computationally hard problems;
3. using them as a modeling platform.

In relation with the first item in the previous list an interesting open problem was to find a non-universal model of P systems that induces an infinite hierarchy on the number of membranes.

In [14] several models of P systems answering the problem were proposed and accepted as solutions to it. The models described in [14] (and also in subsequent related research [16,15]) were featured with maximal parallelism, i.e., in each time step the number of performed operations is the maximum number of operations that can be performed.

Here we consider conformon-P systems without maximal parallelism, i.e., in each time step the number of performed operations is any number between 1 and the maximum number of operations that can be performed. We prove that some restricted models of conformon-P systems induce infinite hierarchies on the number of membranes and on the number of input symbols. These results are obtained with deterministic simulations of restricted counter machines.

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 395–408, 2006.

2 Preliminaries

We assume the reader to have familiarity with basic concepts of formal language theory [13], and in particular with the topic of membrane computing [24]. In this section we recall particular aspects relevant to our presentation.

2.1 Counter Automata

Non-rewriting Turing machines were introduced by M.L. Minsky in [21] and then reconsidered in [22] under the name of *program machines*. After their introduction such machines and some variants of them have been studied under different names: in [11] they were called *(multi)counter machines*, in [1] *multipushdown machines*, in [19] *register machines*, and in [12] *counter automata*. Such devices have *counters* (also called registers) each of unbounded capacity recording a natural number or zero.

Simple operations can be performed on the counters: addition of one unit and conditional subtraction of one unit. After each or these operations the machine can change state. The main difference between the original models and some of the subsequent variants indicated above is that the latter may have a read only tape where the input is recorded. In the model introduced by M.L. Minsky, and considered by us, such tape is not present and the input is recorded as a number in one of the counters of the machine. It is shown in [21] that counter automata can simulate any Turing machine (see also [13, Theorem 7.9]).

Formally a counter automaton with n counters ($n \in \mathbb{N}$ set of natural numbers) is defined as $M = (S, R, s_0, f)$, where S is a finite set of *states*, $s_0, f \in S$ are respectively called the *initial* and *final* state; R is the set of *rules* of the form (s_j, l^+, s_n) (if in state s_j increase the value of the counter l of 1 and go to state s_n), or (s_j, l^-, s_i, s_k) (if in state s_j the value of counter l is zero, then go to state s_k; otherwise decrease the value of the counter l by 1 and then go to state s_i).

Configurations and *computations* for counter automata are defined as in [21].

2.2 Conformon-P Systems

In [8], Frisco & Ji introduced a variant of membrane systems called *conformon-P systems* (cP systems). This variant, later studied also in [9,5,6,7], is based on simple and basic concepts inspired by a theoretical model of the living cell centred around *conformon* [17,18].

The concept of conformon was introduced in molecular biology independently in [10] and [26]. The common part of the two definitions is the conformational deformation of (macro) molecules in a cell.

A cP system has conformons, a name-value pair, as objects. If V is an alphabet (a finite set of letters) and $\mathbb{N}_0$ is the set of natural numbers with 0 included, then a conformon is $[\alpha, a]$, where $\alpha \in V$ and $a \in \mathbb{N}_0$, we will say that α is the *name* and a is the *value* of the conformon $[\alpha, a]$. If, for instance, $V = \{A, B, C, \ldots, Z\}$, then $[A, 5], [C, 0], [Z, 14]$ are conformons, while $[AB, 21], [C, -15]$, and $[D, 0.5]$ are not.

Two conformons can interact according to an *interaction rule*. An interaction rule is of the form $r : \alpha \xrightarrow{n} \beta$, where r is the label of the rule, $\alpha, \beta \in V$, and $n \in \mathbb{N}_0$, and it says that a conformon with name α can give n from its value to the value of a conformon having name β. A rule r can be applied only if the value of the conformon with name α is greater than or equal to n. If, for instance, there are conformons $[G, 5]$ and $[R, 9]$ and the rule $r : G \xrightarrow{3} R$, then the application of r leads to $[G, 2]$ and $[R, 12]$ and the rule r cannot be applied to $[G, 2]$.

The compartments (membranes) present in a cP system have a label, every label being different. Compartments can be unidirectionally connected to each other and for each connection there is a *predicate*. A predicate is an element of the set $\{\geq n, \leq n \mid n \in \mathbb{N}_0\}$. Examples of predicates are: $\geq 5, \leq 2$, etc. If, for instance, there are two compartments (with labels) m_1 and m_2 and there is a connection from m_1 to m_2 having predicate ≥ 4, then conformons having value greater than or equal to 4 can pass from m_1 to m_2. In a time unit any number of conformons can move between two connected membranes as long as the predicate on the connection is satisfied. Notice that we have *unidirectional connections* that is: m_1 connected to m_2 does not imply that m_2 is connected to m_1. Moreover, each connection has its own predicate. If, for instance, m_1 is connected to m_2 and m_2 is connected to m_1, the two connections can have different predicates.

Maximal parallelism, i.e., the fact that in each time step the number of performed operations is the maximum number of operations that can be performed, feature present in most of the variants of P systems, is absent in cP systems. In each time step of a cP systems the number of performed operations is any number between 1 and the maximum number of operations that can be performed.

A computation halts when one (any) conformon is present in a specific (acknowledgement) membrane. When this happens no operation is performed even if it could.

2.3 Some Modules for Conformon-P Systems

In the following we will use the concept of *module*: a group of membranes with conformons and interaction rules in a cP system able to perform a specific task.

An example of module is a *splitter* [5]: a module that, when a conformon $[X, x]$ with $x \in \{x_1, \ldots, x_h\}, x_i < x_{i+1},\ 1 \leq i \leq h-1$ is associated with a specific membrane of it, it may pass such a conformon to other specific membranes according to its value x. In Figures 2 and 3 splitters are depicted by a thicker line, their label starts with **spl**, and their edges have '=' as predicate.

Some of the links between membranes present in the cP systems depicted in Figures 2 and 3 have predicates of the kind $[A, a]$ (a conformon). This is a shorthand for a *separator* module [5]: when conformons of type $[X_i, x],\ 1 \leq i \leq h,\ x \geq 1$ are associated with a specific membrane of it, a separator may pass them to specific different membranes according to their name content. So if there is an edge between membrane 1 and membrane 2 having $[A, a]$ as predicate, it means that only the conformons $[A, a]$ can pass from membrane 1 to membrane 2.

The combination of splitters and separators allows us to define *strict interaction* rule: $A^{(\alpha)} \xrightarrow{\gamma} B_{(\beta)}$ where $\alpha, \beta, \gamma \in \mathbb{N}_0$, meaning that a conformon with name A can interact with B passing just γ only if the value of A and B before the interaction is α and β respectively. The detailed module for strict interaction is depicted in Figure 1. Notice that in a strict interaction just γ is passed even if the value of A could be decreased by any multiple of γ.

Similarly interactions of the kind $A \xrightarrow{\gamma} B_{(\beta)}$ and $A^{(\alpha)} \xrightarrow{\gamma} B$ can be defined.

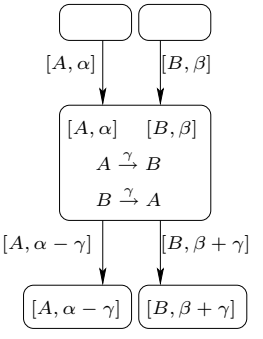


Fig. 1. A detailed strict interaction

In Figures 2 and 3 a forward slash (/) indicates different possibilities for conformons' values, predicates, etc. Moreover, circles with a number indicate membranes having that number as label.

3 Infinite Hierarchies

The search for a non-Turing-complete model of P system for which the number of membranes induces an infinite hierarchy on the computation that can be performed by such system was raised in [24]. Moreover, a prize on the description of such system was advertised in [27].

Candidate solutions to this problem were reported in [4,20], but they were based on definitions that were considered too restrictive, so they were not accepted as solutions.

In [14] several models of P systems answering the problem were proposed and accepted as solutions to it. There *restricted communicating P system* (RCPS) and *restricted counter machines* (RCM) were defined. A RCM is a counter machine which is restricted in its operations: it can increase the value of a counter, say C, only if it decreases the value of another counter, say D at the same time. The counters C and D are said to be *connected*. The research presented in [14] was followed by [16,15] where infinite hierarchies on the number of symbols or on the number of membranes on the computational power of (restricted) variants of P systems were presented. These studies concerned models of P systems with maximal parallelism.

In the following sections we will describe how two variants of P systems without maximal parallelism, basic conformon-P system with restricted features, can induce infinite hierarchies on the computation they can perform. These results are obtained with a deterministic simulation of RCPS.

By a *deterministic simulation* we mean that if the simulated RCM is deterministic, then there will be an isomorphism between the sequence of configurations in the computation of the RCM and some configurations in the basic cP system. This does not mean that in the basic cP system the operations allowing the transition from the simulation of one configuration to the one that deterministically comes after it follow a deterministic path, actually the cP systems defined in the following are non-deterministic. These concepts will be discussed in more details in Section 4.

If the simulated RCM is non-deterministic (one state can be followed by more than one), then in a similar way the simulating basic cP system will be non-deterministic.

3.1 Hierarchy on the Number of Membranes

In this section we consider *conformon-restricted basic cP systems*, i.e., basic cP systems having a conformon with a distinguished name, let us say l, and such that only some membranes (called *input* membranes) contain only l conformons in the initial configuration (this restrictions is equivalent to the one imposed to the RCPSs presented in [14], where only the object o is used to store the value of the simulated RCM).

Conformon-restricted basic cP systems are accepting devices: a computation is a finite sequence of configurations with the initial configuration having some l conformons in the input membranes (and no conformons in the acknowledgement membrane), while the last (*final*) configuration is the only one in the sequence having one (any) conformon in the acknowledgement membrane. As customary in cP systems, when a final configuration is reached no operation is performed even if it could. If for an input there is such a computation, then we say that the conformon-restricted basic cP system *accepts* the input.

Conformon-restricted basic cP systems are equivalent to RCM.

Lemma 1. *Conformon-restricted cP systems can perform a deterministic simulation of a RCM* $M = (S, R, s_0, f)$ *with two (connected) counters.*

Proof. First we explain the general idea, then we will go into the details of the proof. Let us assume that M has two counters: c_1 and c_2 whose content is encoded into the value of one conformon l initially present in, let us say, membrane 5 and membrane 4. We will refer to the former of this conformon with $l_{(m5)}$ and with $l_{(m4)}$ to the latter. Moreover here we concentrate only on the operations performed on the value of $l_{(m5)}$ knowing that the value of $l_{(m4)}$ changes in the opposite way (as the two counters are connected). If the value of counter c_1 is 0 (1), then also the value of $l_{(m5)}$ is 0 (1, respectively); if the value of c_1 is $x > 1$, then the value of $l_{(m5)}$ is $2(x-1)+1$. This means that if the value of c_1 is 0 and it is increased by 1, then the value of $l_{(m5)}$ is increased also

by 1; if the value of c_1 is bigger than 1 and it is increased by 1, then the value of $l_{(m5)}$ is increased by 2. Similarly for a subtraction: if the value of c_1 is 1 and it is decreased (by 1), then the value of $l_{(m5)}$ (is 1 and it) is decreased by 1; in all the other cases the value of $l_{(m5)}$ is decreased by 2.

The cP system is aware of the value of the c_1 counter through some 'state' conformons (defined later on). If the value of the c_1 counter is 0, then the 'state' conformon will carry this information and no further subtraction will be simulated until an addition is performed on that counter. If the value of the c_1 counter is bigger than 0, then subtractions can be simulated.

In this way the number of 'state' conformons is increased by $|S| \times n \times 2$ (where $|S|$ is the number of states and n is the number of counters of M), but the resulting cP system performs a deterministic simulation (and can be computed in polynomial time).

Now we will explain the cP systems in details; during this proof we will refer to Figures 2 and 3.

The initial configuration of the cP system (identified by the conformons in **bold** in Figures 2 and 3) is: for each state n of the simulated RCM there are conformons $[\hat{s}_n, 0]$ and $[\hat{s}_n^{=0}, 0]$ present in membrane 1, $[s'_n, 0]$ in membrane 8, $[\bar{s}_n^{=0}, 0]$ in membrane 20, $[s_n, 0]$ and $[s_n^{=0}, 0]$ in membrane 25, and $[\ddot{s}_n, 0]$ in membrane 29. All these s conformons are called 'state' conformons as they are associated with the states of M. Membranes 4 and 5 (input membranes) contain the conformon with name l whose value reflects the content of the two counters in M (as indicated above). The remaining of the initial configuration is: $[c, 1]$ in membrane 2, $[\bar{c}, 0]$ in membrane 6, $[z, 6]$ in membrane 10, $[k, 0]$ in membrane 16, $[w, 4]$ in membrane 18, and $[v, 1]$ in membrane 22. The rest of the system will be introduced during the description.

As indicated before, the cP system has two conformons associated with each state of M: s_j in case the value of $l_{(m4)}$ is bigger than 0 and $\bar{s}_j^{=0}$ otherwise. Let us assume that the value of $l_{(m4)}$ is bigger than 0 and that M is in state j. In this case $[s_j, 30]$ will be present in membrane 1. If $(s_j, l^+, s_n) \in R$, then the value of conformon $l_{(m4)}$ will be increased by 2 (through the conformon c), the conformon $[s_n, 30]$ is generated using part (2 units) of the value of $l_{(m5)}$ (this is because these two l conformons represent connected counters).

For each rule $(s_j, l^+, s_n) \in R$ the instruction $s_j \stackrel{20}{\rightarrow} \hat{s}_n$ is in membrane 1. In this way the conformons $[\hat{s}_n, 20]$ and $[s_j, 10]$ are generated and they can pass (through spl_1) to membranes 2 and 3 respectively. In membrane 2 $[\hat{s}_n, 20]$ can give (with strict interaction) 2 to the c conformon and then pass to membrane 3 while the c conformon can pass to membrane 5. After passing 2 units to l the c conformon can pass back to membrane 2. In membrane 3 $[\hat{s}_n, 18]$ and $[s_j, 10]$ can interact such that $[s_j, 0]$ and $[\hat{s}_n, 28]$ are generated and they can pass to membrane 25 and 4 respectively. In membrane 4 the value of $\hat{s}_n$ can be increased by 2 (taken from the value of $l_{(m5)}$), when this happens $[\hat{s}_n, 30]$ can pass to membrane 25. Through membranes 25, 27, and spl_3 $[\hat{s}_n, 0]$ and $[s_n, 30]$ are generated and they can pass to membrane 1.

Now we describe how the simulation of one instruction of the kind (s_j, l^+, s_n) in R is performed when the value of the l counter is 0. In this case $[l, 0]$ can be in membrane 13 and no l conformon is present in membrane 5. We will see later on how this happens, for the moment let us take this as a fact. Let us also assume the conformon $[s_j^{=0}, 30]$ is in membrane 1, this simulates that M is in state j. As the instruction $s_j^{=0} \stackrel{21}{\rightarrow} \hat{s}_n$ is also present in this membrane, then the conformons $[s_j^{=0}, 9]$ and $[\hat{s}_n^{=0}, 21]$ can be generated and pass to membranes 14 and 6, respectively (through spl_1). In membrane 6 $[\bar{c}, 2]$ and $[\hat{s}_n^{=0}, 19]$ are generated, afterwards they can pass to membrane 13 and 7, respectively. In membrane 13 $[\bar{c}, 2]$ can give 1 to $[l, 0]$, when this happens $[l, 1]$ can pass to membrane 5, while $[\bar{c}, 1]$ to membrane 7. Notice that in this way the value of l has been increased by 1 (and not 2). When $[\hat{s}_n^{=0}, 19]$ and $[\bar{c}, 1]$ are both in membrane 7 they can interact such that $[\hat{s}_n^{=0}, 20]$ and $[\bar{c}, 0]$ are generated and they can pass to membrane 8 and 6 respectively. When $[\hat{s}_n^{=0}, 20]$ is in membrane 8, then the 'state' conformon $[s_n, 30]$ is generated (indicating that the value of the counter is bigger than 0). This process (similar to what we have described) happens between the membranes 1, 4, 8, 14, 15, 24, 25, 27 and spl_2 and at the end of it $[s_n, 30]$ and $[\hat{s}_n^{=0}, 0]$ are in membrane 1.

Now we describe how an instruction of the kind (s_j, l^-, s_i, s_k) is simulated. In case the 'state' conformon is $s_j^{=0}$, then the counter is empty and the next state is s_k. This is simulated by the rules $s_j^{=0} \stackrel{16}{\rightarrow} \hat{s}_k^{=0}$ present in membrane 1 and allowing $[s_j^{=0}, 14]$ and $[\hat{s}_k^{=0}, 16]$ to be generated. The creation of $[s_k, 30]$ (similar to what described before) is performed through the membranes 25, 32, 33, spl_1, and spl_2.

In case the 'state' conformon is s_j and an instruction of the kind (s_j, l^-, s_i, s_k) is simulated, then two situations are possible. If the value of $l_{(m5)}$ is 1 (indicating that the counter contains 1), then 1 has to be subtracted by $l_{(m5)}$ and the next 'state' conformon has to be $s_i^{=0}$ (indicating that the counter is empty); if the value of $l_{(m5)}$ is bigger than 2 (indicating that the counter contains at least 2), then 2 has to be subtracted by $l_{(m5)}$ and the next 'state' conformon has to be s_i (indicating that the counter is not empty).

If $[s_j, 30]$ is present in membrane 1 and the instruction (s_j, l^-, s_i, s_k) is simulated, then $[s_j, 11]$ and $[\hat{s}_i, 19]$ are generated and they can pass to membranes 23 and 5 respectively. In membrane 5 $\hat{s}_i$ decreases the value of l of 1 (this is always possible) and then $[\hat{s}_i, 20]$ can pass to membrane 16. Here it interacts with $[k, 0]$ so that $[k, 11]$ and $[\hat{s}_i, 9]$ are created and they can pass to membranes 5 and 17 respectively. The role of $[k, 11]$ is to subtract 1 from the value of l. If this is possible then $[k, 12]$ will also pass to membrane 17, if this is not possible then the k conformon will stay in membrane 5 until the z conformon will be also there.

In case after the interaction with $\hat{s}_i$ the value of l becomes 0, then it can pass to membrane 10 and here interact with $[z, 6]$. As a result of this and other interactions (involving membranes 11 and 12) $[l, 0]$ (originally in membrane 4) can pass to membrane 13 (this fact was considered above), while $[z, 6]$ can pass

to membrane 17. It should be clear now that in membrane 17 either $[k, 12]$ or $[z, 6]$ is present, they cannot be present together.

If $[k, 12]$ is present, then the 2 units taken from $l_{(m5)}$ are passed to $l_{(m4)}$ and $[s_i, 30]$ is generated and it can pass to membrane 1; if instead $[z, 6]$ is present in membrane 17, then the unit taken from $l_{(m5)}$ is passed to $l_{(m4)}$ and $[s_i^{=0}, 30]$ is generated and it can pass to membrane 1. These processes, similar to others described above, happen in between the membranes 17-33 (with the exception of membrane 26).

If j is a final state for M, then either $[s_j, 30]$ or $[s_j^{=0}, 30]$ will be present in membrane 1. When this happens, then the application of either $s_j \stackrel{18}{\rightarrow} \hat{s}_f$ or $s_j^{=0} \stackrel{18}{\rightarrow} \hat{s}_f$ will create a conformon with value 18. This conformon can pass to membrane 26 (the acknowledgement membrane) halting in this way the computation. □

The just given constructive proof can be used to create conformon-restricted cP systems that can perform a deterministic simulation of RCMs (with any number of connected counters).

Let us assume that a specific RCM has m counters $C = \{c_1, \ldots, c_m\}$ each with an initial value. Then it is possible to build a conformon-restricted cP systems Π' having l conformons present in m different input membranes. Considering the proof of Lemma 1 this seems to be a must as collecting more than one conformon with name l in the same membrane would not allow the system Π' to perform a simulation on the RCM (we will discuss this point in Section 4). The system Π' would be such that every time the value of an l conformon is increased (decreased), then the one of its connected counter (for the particular simulated instruction) is decreased (increased, respectively) by the same amount. The information on the connected counter can be present in the name or in the value of the 'state' conformons (similarly to what was done in the previous proof).

So we can state:

Corollary 1. *Conformon-restricted cP systems can perform a deterministic simulation of RCMs.*

Here is the reverse of this inclusion:

Lemma 2. *A RCM with m counters can simulate a conformon-restricted cP system having m input membranes.*

Proof. In the initial configuration the value stored in the m counters is the value of the l conformons present in the initial configuration of the cP system. The rest of the description of the cP system is encoded into the finite control of the RCM.

The increase/decrease of the value of the l conformons is split into a sequence of increases/decreases of 1. Every time 1 is subtracted by the value of an l conformon, then the value of the associated counter is decreased by 1 and the one of the connected counter is increased by 1. Chains of coupled increase/decrease of connected counters simulate the passage of value between different conformons. Similarly when the value of an l conformon is increased.

When the simulation of a conformon passing to the acknowledgement membrane is simulated, then the RCM goes into a final state. □

The fact that RCMs induce an infinite hierarchy on the number of counters is proved in [14]. Considering this, we can state:

Theorem 1. *Conformon-restricted cP systems induce an infinite hierarchy on the number of membranes.*

3.2 Hierarchy on the Number of Symbols

In this section we consider *membrane-restricted basic cP systems*, i.e., basic cP systems in which the number of input membranes is restricted to one and the set of names of input conformons is bounded.

An initial configuration is such that some input conformons are present in the input membrane, no input conformon is present in the remaining membranes, and the acknowledgement membrane is empty. We say that a membrane-restricted basic cP systems *accepts* an input if there is a finite sequence of configurations starting from an initial configuration and ending with a (*final*) configuration (being the last one in the sequence) in which one (any) conformon is in the acknowledgement membrane. As customary in cP systems, when a final configuration is reached no operation is performed even if it could.

Lemma 3. *Membrane-restricted cP systems with two input conformons can perform a deterministic simulation of a RCM $M = (S, R, s_0, f)$ with two connected counters.*

Proof. (sketch) The proof follows the one of Lemma 1. Let us assume that the names of the input conformons are l_1 and l_2 and that their initial value reflects the initial content of the counters in M (in the same way as it is done in the proof of Lemma 1).

The operations performed on l_1 and l_2 are similar to the ones performed to the l conformons in the proof of Lemma 1.

The simulation of an instruction of the RCM changing its state into a final one lets a conformon to go into the acknowledgement membrane halting in this way the computation. □

Let us assume that a specific RCM has m counters $C = \{c_1, \ldots, c_m\}$, then it is possible to build a membrane-restricted cP system having input conformons with name $c_1, \ldots, c_m$ in the input membrane. The equivalence between the number of counters and the number of different names of input conformons seems to be a must (we will discuss this point in Section 4). The membrane-restricted cP system would be such that every time the value of an input conformon is increased (decreased), then the one of its connected counter (for the particular simulated instruction) is decreased (increased, respectively) by the same amount. The information on the connected counter can be present in the name or in the value of the 'state' conformons.

So we can say:

Corollary 2. *Membrane-restricted cP systems can perform a deterministic simulation of RCMs.*

Here is the reverse of this inclusion whose proof follows the one of Lemma 2:

Lemma 4. *A RCM with m counters can simulate a membrane-restricted cP system having m input conformons.*

If we now consider that RCMs induce an infinite hierarchy on the number of counters [14], then we have:

Theorem 2. *Membrane-restricted cP systems induce an infinite hierarchy on the number of input conformons.*

4 Final Remarks

At the beginning of Section 3 we defined *deterministic simulation* and we indicated that the restricted cP systems used by us are non-deterministic even if they can perform deterministic simulations.

The non-determinism present in the considered cP systems arises from the fact that some operations can be performed in parallel. If, for instance, the operations A and B can be performed in parallel in a cP system, then (as maximal parallelism is not present) A can be applied before B, or B can be applied before A, or A and B can be applied at the same time.

We are going to investigate if it is possible to have the results presented in this paper with deterministic cP systems. In this respect we will consider to use Petri nets [25] as done in [6].

It is also relevant to notice that the use of separator modules on the conformons representing counters let the sum of these conformons not to be constant but to fluctuate. This fluctuation is due to the separator module used by (variants of) strict interactions (see Figure 1 and [5, Figure 2]). Because of the way the cP systems described in this paper have been devised, these fluctuations do not interfere with the simulations performed by them.

An essential element in the proof of Lemma 1 is the presence of connected loops. Here with *loop* we mean that, during the simulation, some name of conformons cycle in between some membranes (the conformon itself changes because of its value). In the proof of Lemma 1 the k conformon loops between membranes 16, 5, and 17 or between membranes 16, and $(5, 9)^+$ (the superscript $^+$ indicates that k can keep passing between membranes 5 and 9), while the z conformon loops between membranes 10, 11, 12, 17, 5, 9, and 19.

Two loops are *connected* if one can be completed only if the other is taking place. In the proof of Lemma 1, for instance, the k conformon can complete the 16, $(5, 9)^+$ loop only when the z conformon is traversing its loop.

We think that loops are necessary in a system lacking maximal parallelism in order to have some computational power. We also think that the number of loops and their connections can be a measure of computational complexity of system lacking maximal parallelism. The concepts of (connected) loops seems

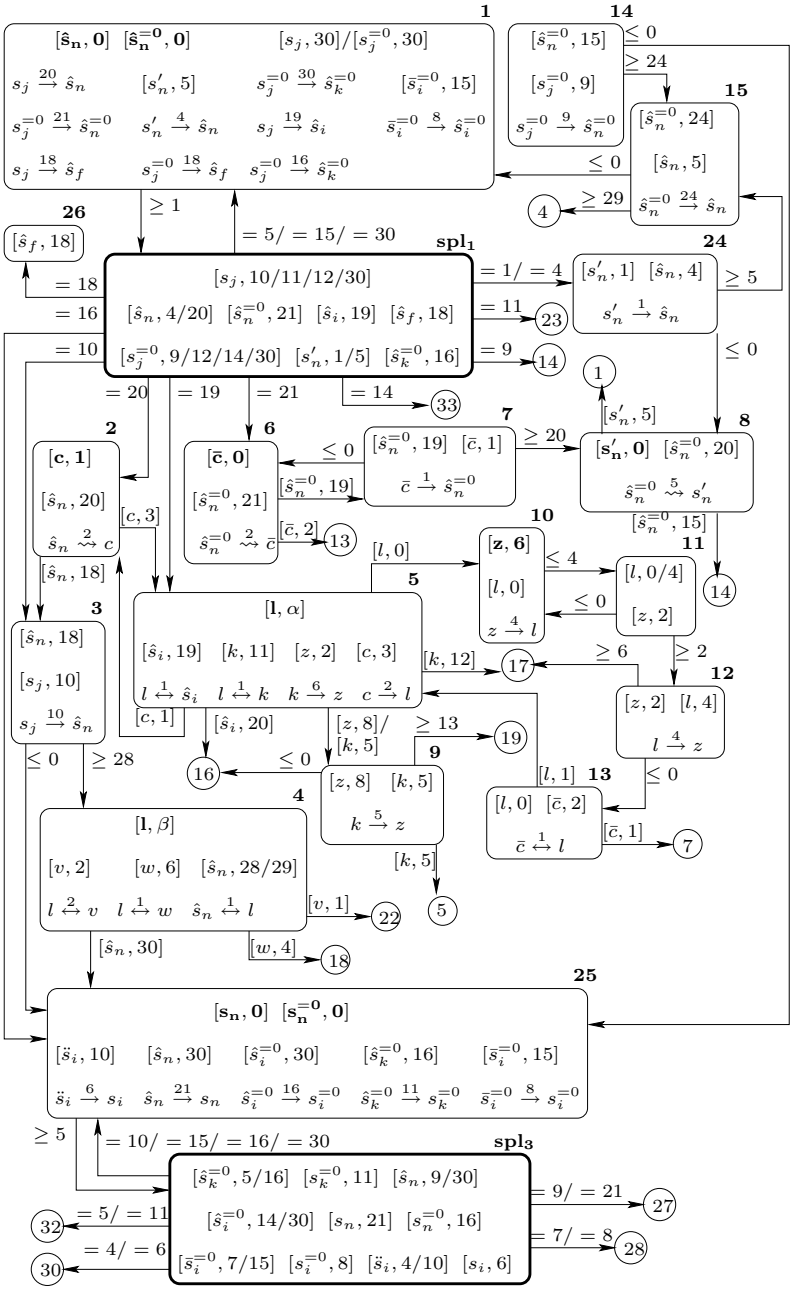


Fig. 2. Conformon-restricted cP systems associate to Lemma 1 (part 1)

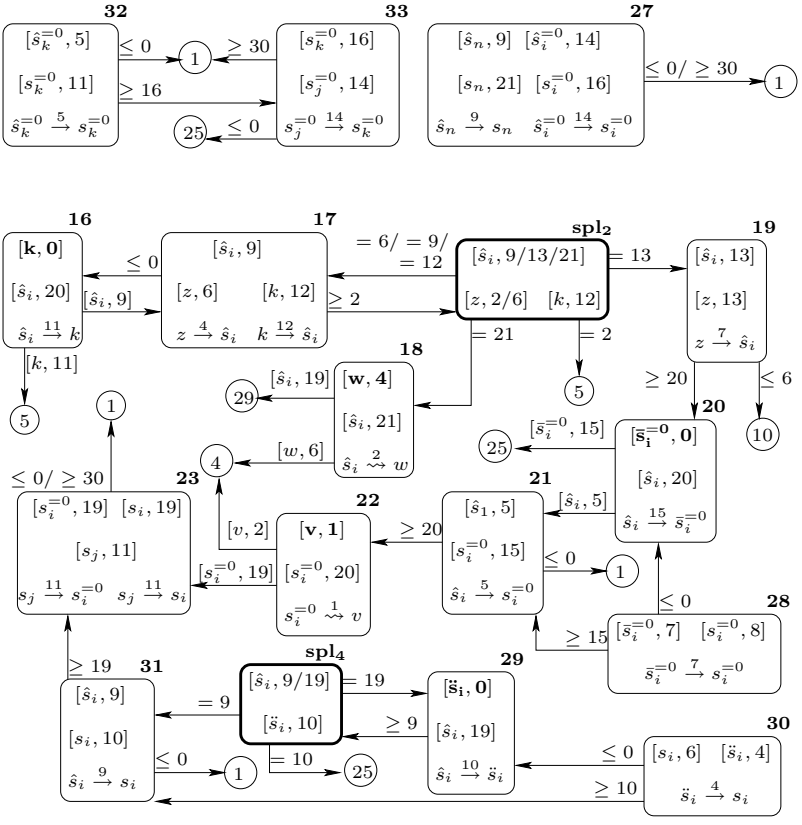


Fig. 3. Conformon-restricted cP systems associate to Lemma 1 (part 2)

to us to be related to the ones of concurrent steps and subsystem in Petri nets [25] and to the one of cycles and topology in directed graphs [2,3]. This reminds us of several biological processes that can be modeled with cyclic directed graphs: metabolic pathways, signalling pathways, gene regulatory networks, etc. We count to investigate this further.

The deterministic simulation can be also used to obtain other results. In [5] Theorem 3 states that a cP system with unbounded total value can perform a (non-deterministic) simulation of a program machine (meaning that in that proof 0-gamble – see [6] – is present). Considering the proof of Lemma 1 we can then state:

Theorem 3. *Conformon-P systems with unbounded value can perform a deterministic simulation of program machines.*

We conclude this paper posing a problem related to variants of P systems inducing an infinite hierarchy (so, also related to what reported in [14,15,16]). The proofs on the presence of such hierarchies are based on a mapping from the space of configurations of the simulated system (for instance, an RCM) to the

considered P system model. In this mapping the content of the counters of the RCM is represented, for instance, by the number of objects in the P system.

Is it possible to have another mapping such that the infinite hierarchy is not induced?

In Section 3.1 we wrote: "... this seems to be a must as collecting more than one conformon with name l in the same membrane would not allow the system Π' to perform a simulation on the RCM.".

This 'must' has no proof (we were not able to give one), even if our common sense suggests that it is not possible to do otherwise.

References

1. B. S. Baker and R. V. Book. Reversal-bounded multipushdown machines. *Journal of Computer and System Science*, 8:315–332, 1974.
2. B. Bollobás. *Extremal graph theory.* Academic Press, 1978.
3. G. Chartrand and L. Lesniak. *Graphs and Digraphs.* Chapman and Hall/CRC, 2005.
4. R. Freund. Special variants of P systems inducing an infinite hierarchy with respect to the number of membranes. *Bulletin of EATCS*, 75:209–219, 2001.
5. P. Frisco. The conformon-P system: A molecular and cell biology-inspired computability model. *Theoretical Computer Science*, 312(2-3):295–319, 2004.
6. P. Frisco. P systems, Petri nets, and Program machines. In R. Freund, G. Lojka, M. Oswald, and G. Păun, editors, *Proceedings 6^{th} International Workshop on Membrane Computing (WMC6)*, volume 3850 of *Lecture Notes in Computer Science*, pages 209–223. Springer-Verlag, Berlin, Heidelberg, New York, 2006.
7. P. Frisco and R. T. Gibson. A simulator and an evolution program for conformon-P systems. In *SYNASC 2005, 7^{th} International Symposium on Simbolic and Numeric Algorithms for Scientific Computing*, pages 427–430. IEEE Computer Society, 2005. Workshop on Theory and Applications of P Systems, TAPS, Timisoara (Romania), September 26-27, 2005.
8. P. Frisco and S. Ji. Conformons-P systems. In M. Hagiya and A. Ohuchi, editors, *DNA8, 8^{th} International Meeting on DNA Based Computers, Hokkaido University, Sapporo, Japan, June 10-13*, volume 2568 of *Lecture Notes in Computer Science*, pages 291–301, 2002.
9. P. Frisco and S. Ji. Towards a hierarchy of info-energy P systems. volume 2597 of *Lecture Notes in Computer Science*, pages 302–318. Springer-Verlag, Berlin, Heidelberg, New York, 2002.
10. D. E. Green and S. Ji. The electromechanical model of mitochondrial structure and function. *Molecular Basis of Electron Transport*, pages 1–44, 1972. J. Schultz, B. F. Cameron (eds).
11. S. A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7:311–324, 1978.
12. H. J. Hoogeboom. Carriers and counters. P-systems with carriers vs. (blind) counter automata. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Developments in language theory, 6^{th} International conference, DLT 2002, Kyoto, Japan, September 2002, Revised papers*, volume 2450 of *Lecture Notes in Computer Science*, pages 140–151, 2003.
13. J. E. Hopcroft and D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

14. O. H. Ibarra. On membrane hierarchy in P systems. *Theoretical Computer Science*, 334:115–129, 2005.
15. O. H. Ibarra and G. Păun. Characterizations of context-sensitive languages and other languages classes in terms of symport/antiport P systems. *Theoretical Computer Science*, 2006. in press.
16. O. H. Ibarra and S. Woodworth. On bounded symport/antiport P systems. In *Preproceedings of The 11th International Meeting on DNA Computing*, pages 25–36, 2005. London, Ontario, Canada.
17. S. Ji. The Bhopalator: a molecular model of the living cell based on the concepts of conformons and dissipative structures. *Journal of Theoretical Biology*, 116:395–426, 1985.
18. S. Ji. The Bhopalator: an information/energy dual model of the living cell (II). *Fundamenta Informaticae*, 49(1-3):147–165, 2002.
19. J. P. Jones and Y. V. Matijasevič. Register machine proof of the theorem on exponential Diophantine representation of enumerable sets. *Journal of Symbolic Logic*, 49(3):818–829, September 1984.
20. S. Krishna. *Languages of P systems: computability and complexity*. PhD thesis, Indian instituto of technology Madras, 2001. India.
21. M. L. Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, 1961.
22. M. L. Minsky. *Computation: Finite and Infinite Machines*. Automatic computation. Prentice-Hall, 1967.
23. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 1(61):108–143, 2000.
24. G. Păun. *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, Heidelberg, New York, 2002.
25. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, New York, 1998.
26. M. V. Volkenstein. The conformon. *Journal of Theoretical Biology*, 34:193–195, 1972.
27. C. Zandron. P-systems web page: http://psystems.disco.unimib.it.

A Protein Substructure Based P System for Description and Analysis of Cell Signalling Networks

Thomas Hinze, Thorsten Lenser, and Peter Dittrich

Friedrich Schiller University Jena
Bio Systems Analysis Group
Ernst-Abbe-Platz 1–4, D-07743 Jena, Germany
{hinze,thlenser,dittrich}@minet.uni-jena.de

Abstract. The way how cell signals are generated, encoded, transferred, modified, and utilized is essential for understanding information processing inside living organisms. The tremendously growing biological knowledge about proteins and their interactions draws a more and more detailed image of a complex functional network. Considering signalling networks as computing devices, the detection of structural principles, especially modularization into subunits and interfaces between them, can help to seize ideas for their description and analysis. Algebraic models like P systems prove to be appropriate to this. We utilize string-objects to carry information about protein binding domains and their ligands. Embedding these string-objects into a deterministic graph structured P system with dynamical behavior, we introduce a model that can describe cell signalling pathways on a submolecular level. Beyond questions of formal languages, the model facilitates tracing the evolutionary development from single protein components towards functional interacting networks. We exemplify the model by means of the yeast pheromone pathway.

1 Introduction

Protein signalling networks can be viewed as computational devices of the cell, triggering and directing responses to external inputs. Therefore, the utilization of tools and techniques from computer science to study signalling networks constitutes an almost natural step. In this contribution, we propose a specialized version of the P system framework, which is a term-rewriting mechanism designed with cellular principles in mind [12,13].

Cell signalling networks (CSNs) represent a class of biochemical reaction networks, set apart from others (such as metabolic networks) by an arrangement of special properties. One of these is the importance of the configuration of individual molecules (proteins) to their function in the network, which is exemplified by the activation of certain kinases via phosphorylation. Therefore, the protein constituents of CSNs should not be viewed as atomic objects, but rather as

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 409–423, 2006.

entities whose configuration can change over the course of time. Another distinguishing property of CSNs is the importance of their temporal behavior. While the steady-state behavior might be enough to characterize a metabolic network, the function of a CSN depends heavily on its temporal evolution. Thirdly, partitioning the whole pathway into several connected modules is thought to be a keystone for understanding molecular networks, a concept common to both metabolic and signalling systems. The formalism described here intends to incorporate the module concept by altering the established membrane framework from a tree-based to a graph-based structure. Similar approaches to P system structures are considered in tissue and population P systems [2,10].

P systems with string-objects have already been considered in [12], while graph-based membrane structures were introduced in [14], and dynamics in [15,18]. Our approach to temporal dynamics is based on the metabolic algorithm developed in [8], where a kinetic understanding of P system rewriting rules is explained and simulated. While these features have been introduced and investigated previously, the novelty of the approach presented here lies in their combination into one system, in order to define a framework suitable for modeling complex CSNs. A detailed account of the system is provided, and its suitability for modeling CSNs is demonstrated by an extensive example model of the yeast pheromone pathway.

2 Modeling Cell Signalling Networks

Modern molecular biology yields more and more data which shed light on signalling processes in the cell. In order to keep up with these developments, theoretical biologists and computer scientists have to provide modeling formalisms capable of integrating this growing knowledge. This section will briefly review common practices in modeling CSNs and introduce the advantages of using the P systems approach for this task, together with the extensions and novelties that are proposed in this contribution.

The most common class of formal CSN models consists of analytical models, based on differential equations. These systems are relatively straightforward to set up from a reaction network, and a plethora of tools for their numerical evaluation is available. Unfortunately, differential equations do not allow to include much human-readable information into the model, so that large models quickly become incomprehensible. Additionally, due to their continuous nature, these approaches are usually not tractable by tools and theory of computer science.

A classic step from continuous to discrete models leads to stochastic approaches, in which the number of molecules of each molecular species is assumed to be a random variable with temporally varying probability distribution. The conversion of a set of reactions into a stochastic system description has been addressed by a range of publications, mostly building on the fundamental work by Gillespie [5].

Algebraic approaches, drawing heavily on concepts from theoretical computer science, have several advantages to offer. They are well understood, cover a wide

field of different modeling aspects, and each one comes with its own set of tools to analyze specific models. Additionally, they enable structuring and classification on several levels of abstraction. A short and by no means complete list of such formalisms would include: state-based systems such as abstract machines and X machines [4], process calculi such as π calculus [11], ambient calculus [3] and Petri nets [16], and term-rewriting systems based on Chomsky-grammars [17]. P systems, which represent an instance of the last category, are especially suited to develop models of cellular computation.

The general P system framework [13] is based on rewriting of multisets of molecule objects, which are contained in different compartments of the cell. Rewriting rules are localized to the compartments, so that object-processing depends on the current localization. Objects can move between compartments, allowing the flow of a signal through the system. Here, we extend this formalism with a set of concepts that are essential for modeling CSNs.

By considering string-objects, we allow the substructures and properties of individual proteins to carry information. Extending the original concept of membranes to the more abstract view of distinguished modules in the CSN leads to models built out of coherent components, which are easier to create, maintain and re-use. In order to enable detailed studies on the temporal evolution of the system, we replace the maximally parallel rewriting from the original framework with a mechanism that is based on reaction kinetics. For each rewriting rule, the number of applications per turn is given by a kinetic function, depending on the current configuration of the module. This way, a deterministic system evolution is obtained.

3 System Description

We introduce a deterministic rewriting P system based on multisets of string-objects. The system description combines aspects of formal languages with numeric evaluations for handling of object selection and multiplicities. String-objects are composed in a way to encode information about protein substructures and specific protein properties.

Formal Language Prerequisites

We denote the empty word by ε. The concatenation of formal languages L_1 and L_2 over a common alphabet Σ is written as $L_1 \otimes L_2 = \{uv \mid u \in L_1 \wedge v \in L_2\}$. $\mathcal{P}(L)$ denotes the power set of L. Let A be an arbitrary set and $\mathbb{N}$ the set of natural numbers including zero. A multiset over A is a mapping $F : A \longrightarrow \mathbb{N} \cup \{\infty\}$. $F(a)$, also denoted as $[a]_F$, specifies the multiplicity of $a \in A$ in F. Multisets can be written as an elementwise enumeration of the form $\{(a_1, F(a_1)), (a_2, F(a_2)), \ldots\}$ since $\forall (a, b_1), (a, b_2) \in F : b_1 = b_2$. The support $\mathrm{supp}(F) \subseteq A$ of F is defined by $\mathrm{supp}(F) = \{a \in A \mid F(a) > 0\}$. A multiset F over A is said to be empty iff $\forall a \in A : F(a) = 0$. The cardinality $|F|$ of F over A is $|F| = \sum_{a \in A} F(a)$. Let F_1 and F_2 be multisets over A. F_1 is a subset of F_2, denoted as $F_1 \subseteq F_2$, iff $\forall a \in A : (F_1(a) \leq F_2(a))$. Multisets F_1 and F_2

are equal iff $F_1 \subseteq F_2 \wedge F_2 \subseteq F_1$. The intersection $F_1 \cap F_2 = \{(a, F(a)) \mid a \in A \wedge F(a) = \min(F_1(a), F_2(a))\}$, the multiset sum $F_1 \uplus F_2 = \{(a, F(a)) \mid a \in A \wedge F(a) = F_1(a) + F_2(a)\}$, and the multiset difference $F_1 \ominus F_2 = \{(a, F(a)) \mid a \in A \wedge F(a) = \max(F_1(a) - F_2(a), 0)\}$ form multiset operations. Multiplication of a multiset $F = \{(a, F(a)) \mid a \in A\}$ with a scalar c, denoted $\mathrm{c} \cdot F$, is defined by $\{(a, \mathrm{c} \cdot F(a)) \mid a \in A\}$. The term $\langle A \rangle = \{F : A \longrightarrow \mathbb{N} \cup \{\infty\}\}$ describes the set of all multisets over A.

Definition of the System

Let $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$ be the set of natural numbers without zero. A P system for describing CSNs of degree $n \in \mathbb{N}_+$ is a construct

$$\Pi_{\mathrm{CSN}} = (V, V', E, M, n)$$

where V and V' are two alphabets; without loss of generality $\#, \neg, * \notin V \cup V'$. Furthermore, E and M specify channels and modules. The regular set

$$S = V^+ \otimes \left(\{\#\} \otimes ((V')^+ \cup \{\neg\} \otimes (V')^+ \cup \{*\})\right)^*$$

describes the syntax for string-objects. The leftmost substring from V^+ holds the protein identifier, followed by a finite number of protein property substrings from $(V')^+$ which are separated by #. For example, consider the string-object C:D#p#*#¬q identifying protein (complex) C:D with specified property p, a second arbitrary property (*), and without property q. Each protein property substring expresses a specific additional information about the protein, for instance whether it is activated with respect to a certain function or carries a ligand at a certain binding site. Two kinds of meta symbols are allowed. The symbol ¬ excludes the subsequent property but permits all other properties at this substring position. The placeholder * stands for an arbitrary (also unknown or unspecified) protein property substring. This way, uncertainty about the properties of proteins can be explicitly expressed. String-objects can be processed inside modules and they can move between modules along predefined channels (edges). The finite set of modules

$$M = \{M_1, \ldots, M_n\}$$

defines functional reaction units where multisets of string-objects can be modified by regulated rewriting. A module need not be embedded into a physical membrane, it just represents a space where reactions can occur. Multiple modules are allowed to share the same physical space. Modules are intended to form small units that fulfill well-defined functions. Each module M_i is defined as a tuple:

$$\begin{aligned}
M_i &= (R_{i1}, \ldots, R_{ir_i}, \mathrm{f}_{i1}, \ldots, \mathrm{f}_{ir_i}, A_i) \quad \text{where} \\
R_{ij} &\in \langle S \rangle \times \langle S \rangle \quad \text{is a reaction rule composed of two finite multisets} \\
\mathrm{f}_{ij} &: \langle S \rangle \longrightarrow \mathbb{N} \quad \text{is a function corresponding to kinetics of reaction } R_{ij} \\
A_i &\in \langle S \rangle \quad \text{is a multiset of axioms representing the initial contents of } M_i
\end{aligned}$$

The set of channels is defined as

$$E \subseteq \{1,\ldots,n\} \times \{1,\ldots,n\} \times \mathcal{P}(S \times \{\mathrm{g} : \langle S \rangle^2 \longrightarrow \mathbb{N}\}) \times \mathbb{N} \quad \text{where}$$
$$e_{ij} = (i, j, I_{ij}, \mathrm{d}_{ij}) \in E$$

represents a directed channel from module M_i to module M_j. String-objects are allowed to pass the channel if they match the filter interface denoted by the construct

$$I_{ij} \subseteq \{(w, \mathrm{g}_{w,ij}) \mid w \in S \wedge \mathrm{g}_{w,ij} : \langle S \rangle^2 \longrightarrow \mathbb{N}\}.$$

The elements of I_{ij} correspond to the notion of filter patterns (receptors) w and concentration gradients $\mathrm{g}_{w,ij}$ between source module M_i and destination module M_j. Function $\mathrm{g}_{w,ij}$ marks the maximum capacity of the channel for string-objects matching the pattern w, depending on the contents of M_i and M_j. For simplicity, we assume that all filter interface patterns of channels beginning at the same module are pairwise disjoint to each other: $\bigcap_{j \in \{1,\ldots,n\}} \mathit{Match}(\mathrm{supp}(I_{ij})) = \emptyset \; \forall i \in \{1,\ldots,n\}$ where $\mathit{Match} : \mathcal{P}(S) \longrightarrow \mathcal{P}(S)$ is defined in the next subsection. The support of the construct I_{ij} is defined in analogy to multisets. The natural number d_{ij} attached to each channel defines its time delay. Each passing string-object takes this amount of time when moving from module M_i to module M_j.

Matching and Matching Strategies

String-objects may contain excluding symbols $\neg$ and wild-cards $*$ to express partially incomplete knowledge about protein properties. Selecting string-objects for reactions and deciding which string-objects are allowed to pass a channel requires a definition of matching. Matching evaluates whether or not string-objects fit to each other, considering their identifiers and all possible combinations of protein property substrings resulting from their wild-carded patterns. We can distinguish between several matching strategies that differ by their handling of uncertainty. Extreme versions of matching are characterized by a loose and a strict strategy. A prerequisite of matching string-objects is their common number of property substrings.

In the symmetric relation $\mathit{Match}_{\mathrm{loose}}$, two string-objects match iff there is at least one common wild-card free representation. The loose strategy requires a minimum degree of similarity between objects with incomplete information. Uncertainty is interpreted as arbitrary replacements within the search space given by S.

$$\mathit{Match}_{\mathrm{loose}} \subseteq S \times S$$
$$\mathit{Match}_{\mathrm{loose}} = \bigcup_{m \in \mathbb{N}} \{(p\#p_1\#p_2 \ldots \#p_m,\ s\#s_1\#s_2 \ldots \#s_m) \mid (p = s) \wedge$$
$$\forall j \in \{1,\ldots,m\} : [(p_j = s_j) \vee (p_j = *) \vee (s_j = *) \vee$$
$$((p_j = \neg q) \wedge (s_j \neq q)) \vee ((s_j = \neg q) \wedge (p_j \neq q))]\}$$

In contrast, the strict matching strategy follows the opposite intention. The two participating string-objects are interpreted as a pattern and a candidate for matching. Matching only occurs when the candidate $s\#s_1\#s_2 \ldots \#s_m$ is a concretion of the pattern $p\#p_1\#p_2 \ldots \#p_m$. The strict strategy embodies a matching with maximum degree of similarity between string-objects. Because of the different roles of the matching partners, the strict matching relation is not necessarily symmetric.

$$
\begin{aligned}
&Match_{\text{strict}} \subseteq S \times S \\
&Match_{\text{strict}} = \bigcup_{m \in \mathbb{N}} \{(p\#p_1\#p_2 \ldots \#p_m,\ s\#s_1\#s_2 \ldots \#s_m) \mid (p = s) \wedge \\
&\qquad \forall j \in \{1, \ldots, m\} : [(p_j = s_j) \vee (p_j = *) \vee ((p_j = \neg q) \wedge (s_j \neq q))]\}
\end{aligned}
$$

Let the regular set S be a syntax description for string-objects. Matching of a single string-object $w \in S$ to the search space generated by S is defined by

$$Match(w) = \{s \in S \mid (w, s) \in Match_{\text{x}}\}$$

with x = loose or x = strict. Consequently, we define the matching of a language $L \subseteq S$ by the function $Match : \mathcal{P}(S) \longrightarrow \mathcal{P}(S)$ with

$$Match(L) = \bigcup_{w \in L} Match(w).$$

Definition of System Behavior

This subsection describes the dynamical behavior of P systems Π_{CSN}. The multiset $L_i(t)$ denotes the contents of module M_i at time $t \in \mathbb{N}$. $L_i(t)$ is assumed to be empty for $t < 0$. It represents the configuration of the module controlled by a global clock and leads to the definition of the system step:

$$
\begin{aligned}
L_i(0) &= A_i \\
L'_i(t) &= L_i(t) \ominus Educts_i(t) \uplus Products_i(t) \\
L_i(t+1) &= L'_i(t) \ominus Outgoing_i(t) \uplus Incoming_i(t)
\end{aligned}
$$

A system step consists of four stages of modification, each of which is carried out synchronously in all modules. Firstly, the multiset of reaction educts is determined and removed from the module contents $L_i(t)$. Controlled application of local reaction rules transforms these educts into a multiset of products, which is added to the module contents without time delay. A subset of the new module contents can enter outgoing channels to move to (other) modules. Finally, arriving string-objects that have passed channels towards the module complete its contents.

Let $R_{ij} = (F_A, F_B) \in \langle S \rangle \times \langle S \rangle$ be a reaction rule in module M_i with $\text{supp}(F_A) = \{a_1, \ldots, a_p\}$ and $\text{supp}(F_B) = \{b_1, \ldots, b_q\}$. In terms of a chemical denotation, the rule R_{ij} can be written as

$$F_A(a_1) \cdot a_1 + \ldots + F_A(a_p) \cdot a_p \longrightarrow F_B(b_1) \cdot b_1 + \ldots + F_B(b_q) \cdot b_q$$

where $F_A(a_1), \ldots, F_A(a_p)$ encode stoichiometric factors of educts $a_1, \ldots, a_p$, and $F_B(b_1), \ldots, F_B(b_q)$ stoichiometric factors of products $b_1, \ldots, b_q$, respectively. All educt strings that match to the pattern a_k are provided by $Match(a_k)$. A combination of educt strings from $L_i(t)$ matching the left hand side of R_{ij} forms a multiset of string-objects used to apply the reaction once. Since the kinetic law, described by the scalar function f_{ij}, returns the number of applications of reaction rule R_{ij} within one step, the multiset of string-objects extracted from $L_i(t)$ to act as educts for R_{ij} can be written as $Educts_{ij}(t)$:

$$Educts_{ij}(t) = \biguplus_{e_1 \in Match(a_1)} \ldots \biguplus_{e_p \in Match(a_p)} \mathrm{f}_{ij}\big(\{(e_1,\infty),\ldots,(e_p,\infty)\} \cap L_i(t)\big) \cdot \big\{(e_1, F_{A_{ij}}(a_1)),\ldots,(e_p, F_{A_{ij}}(a_p))\big\}$$

Considering educts of all reaction rules $R_{i1}, \ldots, R_{ir_i}$ in module M_i, we achieve

$$Educts_i(t) = \biguplus_{j \in \{1,\ldots,r_i\}} Educts_{ij}(t).$$

Equivalently, the multiset of products obtained from reaction rule R_{ij} is determined by the multiset $Products_{ij}(t)$:

$$Products_{ij}(t) = \biguplus_{e_1 \in Match(a_1)} \ldots \biguplus_{e_p \in Match(a_p)} \mathrm{f}_{ij}\big(\{(e_1,\infty),\ldots,(e_p,\infty)\} \cap L_i(t)\big) \cdot \big\{(b_1, F_{B_{ij}}(b_1)),\ldots,(b_q, F_{B_{ij}}(b_q))\big\}$$

Considering products of all reaction rules $R_{i1}, \ldots, R_{ir_i}$ in module M_i, we achieve

$$Products_i(t) = \biguplus_{j \in \{1,\ldots,r_i\}} Products_{ij}(t).$$

Although the multiset difference always returns non-negative multiplicities, also in case of a lack of educt-objects, the number of product-objects is only determined by B_{ij}. This effect could be compensated by extension of multiset multiplicities to negative integers as well. This way, the requirement of mass-balance could formally be sustained without additional formalism. For sufficiently large numbers of proteins, however, this effect is negligible.

After performing the reactions, the multisets of outgoing and incoming string-objects are specified using $L'_i(t)$ and filter interfaces I_{ij}. Let

$$Outgoing_{ij}(t) = L'_i(t) \cap \big\{(v, \mathrm{g}_{w,ij}(L'_i(t), L'_j(t))) \,\big|\, v \in S \wedge w \in \mathrm{supp}(I_{ij}) \wedge v \in Match(w)\big\}$$

the multiset of transferred string-objects along the channel from M_i to M_j. We define:

$$Outgoing_i(t) = \biguplus_{j \in \{1,\ldots,n\}} Outgoing_{ij}(t)$$

$$Incoming_i(t) = \biguplus_{k \in \{1,\ldots,n\}} Outgoing_{ki}(t - \mathrm{d}_{ki})$$

Generated Language

This subsection specifies the configuration of system Π_{CSN} at time t and finally the generated formal language $L(\Pi_{\mathrm{CSN}})$. The contents $L_i(t)$ of all modules M_i form the essential part of the system configuration. Since string-objects can take several time steps to pass channels, the system configuration at time t also subsumes all multisets $Outgoing_{ij}(\tau)$ with $\tau = 0, \ldots, t-1$. The configuration of module M_i at time t is a construct

$$C_i(t) = \left(L_i(t), \left(Outgoing_{ij}(\tau) \right)_{\substack{j=1,\ldots,n \\ \tau=0,\ldots,t-1}} \right).$$

Furthermore,

$$C_{\Pi_{\mathrm{CSN}}}(t) = (C_1(t), \ldots, C_n(t)).$$

Π_{CSN} generates the language

$$L(\Pi_{\mathrm{CSN}}) = \mathrm{supp}\left(\biguplus_{t=0}^{\infty} \left(\biguplus_{i=1}^{n} L_i(t) \right) \right),$$

the set of string-objects that occur in any module during infinite execution of the system.

4 System Properties

P systems of the framework Π_{CSN} feature a combination of properties which are relevant to describe and analyze CSNs. Bringing together notions of substructured string-objects, configurable modules interconnected by channels, and a formalization of deterministic dynamical system behavior, the proposed approach lends itself to applications beyond classification of computational power. Further studies are focused on the evolutionary development from small low-structured subunits towards much more complex networks with shared resources. Establishing correlations between physical structures and biological functions is a key issue here. In preparation of these objectives, we have designed Π_{CSN} with the following properties.

Modularity: System composition of a finite number of interacting modules follows the idea of defined functional subunits. Modularization can be seen as a powerful tool to represent the inherent structure of a complex system, its organization, and its basic principles. Each module performs a specific set of reactions in an autonomous manner. Communication between modules is separated from reaction processes.

Static System Topology: Established CSNs own a static topology based on modules and directed channels resulting in a graph structure. Each of the channels acts as a filter with regard to both qualitative and quantitative aspects. Configurable patterns represent receptors to accept or reject proteins

with specific properties. Maximum capacity as well as time delay reflect physical restrictions. In CSNs, channels often form cascades consisting of several stages with different protein ligands, complexes or activation state.

Ability to Identify Objects/Substructures: Each single (protein or ligand) molecule handled within the system is treated as an individual object. It identifies the underlaying protein and provides information about additional specific properties. Since reactions within CSNs often keep proteins but modify their properties, consideration of sub-structural information is essential in the model in order to handle combinatorial networks.

Flexibility in Level of Abstraction: The concept of substrings containing information about specific protein properties gives the system a high degree of flexibility in the level of abstraction. Wild-carded and excluding patterns enable coping with uncertainty. The way how incomplete information can be processed by reaction and transduction spans a wide range of detail. Consequences of uncertainty to the system behavior become obvious.

Determinism: The system Π_{CSN} is constructed to work in a deterministic manner. Subsequent execution of system steps leads to a unique path through configurations. In terms of the computational path, determinism implies confluence.

Computational Tractability: Determinism and finite system components facilitate simulations in silico. All aspects of the system description and system behavior are formalized for Π_{CSN}. Sets, multisets, and functions used within the system are polynomially decidable with regard to the number of objects. Software tools like computer algebra systems can serve for further analysis.

Computational Completeness: The ability to use P systems as models for computation can be seen as a fundamental aspect in the field of membrane computing [9]. Investigations about their (sub)classes of computability depending on certain combinations of system properties and restrictions motivate theoretically inspired contributions to the field. Reaction networks are known to be computational complete, constructively shown in [7]. Each module M_i of Π_{CSN} forms such a reaction network.

5 Example: Signal Transduction in the Yeast Pheromone Pathway

The pheromone response pathway in *Saccharomyces cerevisiae* (yeast hereafter) is among the best understood signalling pathways in eukaryotes. Its constituents (proteins) and their interactions have been subject of a great variety of studies, and the overall picture of how these act together in the pathway is rapidly emerging (see [1] for a review). Yeast cells exist in two mating types, *MAT***a** and *MAT*α, which secrete pheromones to stimulate mating behavior in the opposite type. Effects of this stimulation are the arrest of the cell cycle, changes in the

expression of around 200 genes, and even an elongation of the cell in the direction of its mating partner.

To show the suitability of the P system formalism to model cell signalling networks, we have decided to convert the comprehensive yeast pheromone pathway model by Kofahl and Klipp [6] into our framework. The pathway consists of different modules: the G-protein-coupled receptor (M_1, corresponding to receptor activation and G-protein cycle in [6]), formation of the scaffold protein (M_2), MAPK cascade (M_3), Fus3 phosphorylation cycle (M_4), and responses of the cell to the activation of the pathway (not modelled here).

In the module M_1, the receptor protein Ste2 is activated by α-Factor pheromone. In response to activation of Ste2, the trimeric G-protein breaks into its α and $\beta\gamma$ subunits, of which the latter passes on the signal. The "service module" M_2 binds the three components of the MAPK cascade (Ste11, Ste7 and Fus3) to the scaffold protein Ste5, which is then bound by G$\beta\gamma$. Ste20 can now bind to this complex (creating complex C), where it phosphorylates Ste11 and thus triggers the MAPK cascade (M_3). In this cascade, Ste11 activates Ste7, which in turn activates Fus3. Activated Fus3 is then split off (leaving complex C$'$) and moves into the nucleus. Unphosphorylated Fus3 can again bind to C$'$, creating a cycle which amplifies the response.

$$
\begin{aligned}
\Pi_{pheromone} &= (V, V', E, M, 4)\\
V &= \{\mathrm{Ste2}, \alpha, \mathrm{G}\beta\gamma, \mathrm{G}\alpha, \mathrm{Ste5}, \mathrm{Ste11}, \mathrm{Ste7}, \mathrm{Fus3}, \mathrm{Ste20}, \mathrm{C}, \mathrm{C}', :\}\\
V' &= \{\mathrm{a}, \mathrm{GDP}, \mathrm{GTP}, \mathrm{p}\}\\
\\
M &= \{M_1, M_2, M_3, M_4\}\\
E &= \{(1, 3, I_{13}, \mathrm{d}_{13}), (3, 1, I_{31}, \mathrm{d}_{31}), (2, 3, I_{23}, \mathrm{d}_{23}),\\
&\quad (3, 2, I_{32}, \mathrm{d}_{32}), (3, 4, I_{34}, \mathrm{d}_{34})\}\\
\\
I_{13} &= \{(\mathrm{G}\beta\gamma,\ \mathrm{g}_{13}(L'_1(t), L'_3(t)) = \lfloor \mathrm{k}_{\mathrm{g}_{13}}[\mathrm{G}\beta\gamma]_{L'_1(t)} \rfloor)\}\\
I_{31} &= \{(\mathrm{G}\beta\gamma,\ \mathrm{g}_{31}(L'_3(t), L'_1(t)) = \lfloor \mathrm{k}_{\mathrm{g}_{31}}[\mathrm{G}\beta\gamma]_{L'_3(t)} \rfloor)\}\\
I_{23} &= \{(\mathrm{Ste5{:}Ste11{:}Ste7{:}Fus3},\\
&\quad \mathrm{g}_{23}(L'_2(t), L'_3(t)) = \lfloor \mathrm{k}_{\mathrm{g}_{23}}[\mathrm{Ste5{:}Ste11{:}Ste7{:}Fus3}]_{L'_2(t)} \rfloor)\}\\
I_{32} &= \{(\mathrm{Ste5{:}Ste11{:}Ste7{:}Fus3},\\
&\quad \mathrm{g}_{32}(L'_3(t), L'_2(t)) = \lfloor \mathrm{k}_{\mathrm{g}_{32}}[\mathrm{Ste5{:}Ste11{:}Ste7{:}Fus3}]_{L'_3(t)} \rfloor)\}\\
I_{34} &= \{(\mathrm{C\#p\#p\#p\#p},\ \mathrm{g}_{34}(L'_3(t), L'_4(t)) = \lfloor \mathrm{k}_{\mathrm{g}_{34}}[\mathrm{C\#p\#p\#p\#p}]_{L'_3(t)} \rfloor)\}\\
\\
M_1 &= (R_{11}, R_{12}, R_{13}, R_{14}, R_{15}, \mathrm{f}_{11}, \mathrm{f}_{12}, \mathrm{f}_{13}, \mathrm{f}_{14}, \mathrm{f}_{15}, A_1) \quad \text{with}\\
R_{11} &= \mathrm{Ste2\#\neg a} \ +\ \alpha \ \longrightarrow\ \mathrm{Ste2\#a}\\
R_{12} &= \mathrm{Ste2\#a} \ \longrightarrow\ \mathrm{Ste2\#\neg a}\\
R_{13} &= \mathrm{G}\beta\gamma{:}\mathrm{G}\alpha\mathrm{\#GDP} \ +\ \mathrm{Ste2\#a} \ \longrightarrow\ \mathrm{G}\alpha\mathrm{\#GTP} \ +\ \mathrm{G}\beta\gamma \ +\ \mathrm{Ste2\#a}
\end{aligned}
$$

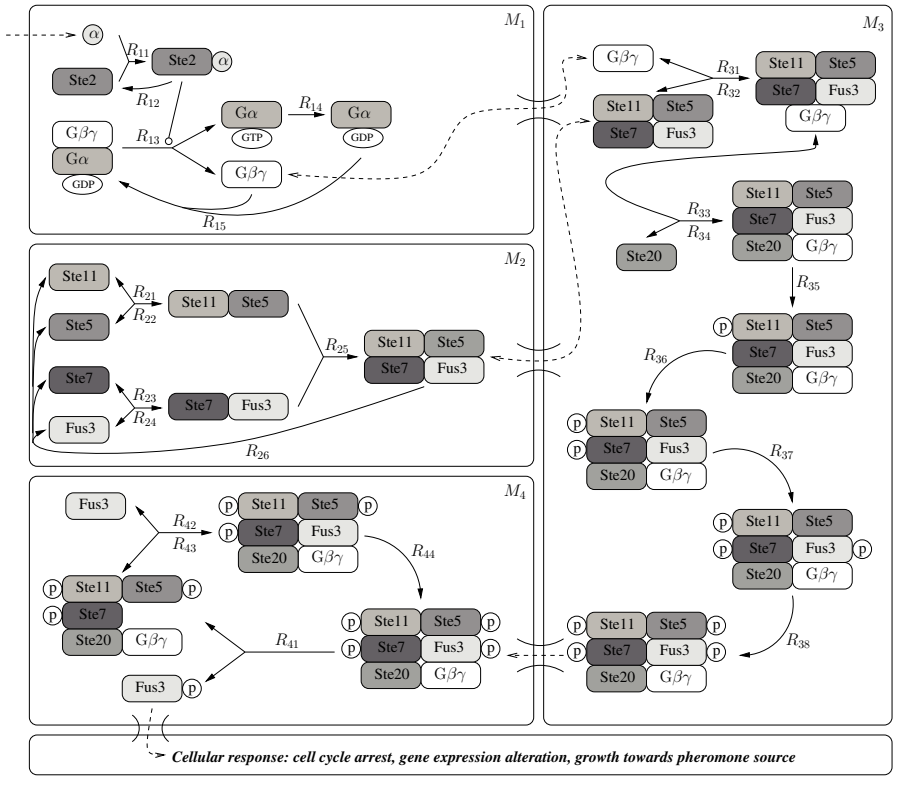


Fig. 1. Module diagram and reaction scheme of the yeast pheromone pathway. At bidirectional arrows, the upper rule corresponds to the rightward direction. Due to the combinatorial complexity, not all sequences of phosphorylation and only one dissociation of complex C are shown in M_3.

$$R_{14} = \mathrm{G}\alpha\#\mathrm{GTP} \longrightarrow \mathrm{G}\alpha\#\mathrm{GDP}$$
$$R_{15} = \mathrm{G}\beta\gamma + \mathrm{G}\alpha\#\mathrm{GDP} \longrightarrow \mathrm{G}\beta\gamma{:}\mathrm{G}\alpha\#\mathrm{GDP}$$
$$\mathrm{f}_{11}(L_1(t)) = \left\lfloor \mathrm{k}_{11}[\mathrm{Ste2}\#\neg\mathrm{a}]_{L_1(t)}[\alpha]_{L_1(t)}(1/\mathrm{V}_1^2) \right\rfloor$$
$$\mathrm{f}_{12}(L_1(t)) = \left\lfloor \mathrm{k}_{12}[\mathrm{Ste2}\#\mathrm{a}]_{L_1(t)}(1/\mathrm{V}_1) \right\rfloor$$
$$\mathrm{f}_{13}(L_1(t)) = \left\lfloor \mathrm{k}_{13}[\mathrm{G}\beta\gamma{:}\mathrm{G}\alpha\#\mathrm{GDP}]_{L_1(t)}[\mathrm{Ste2}\#\mathrm{a}]_{L_1(t)}(1/\mathrm{V}_1^2) \right\rfloor$$
$$\mathrm{f}_{14}(L_1(t)) = \left\lfloor \mathrm{k}_{14}[\mathrm{G}\alpha\#\mathrm{GTP}]_{L_1(t)}(1/\mathrm{V}_1) \right\rfloor$$
$$\mathrm{f}_{15}(L_1(t)) = \left\lfloor \mathrm{k}_{15}[\mathrm{G}\beta\gamma]_{L_1(t)}[\mathrm{G}\alpha\#\mathrm{GTP}]_{L_1(t)}(1/\mathrm{V}_1^2) \right\rfloor$$
$$A_1 = \{(\alpha, 6 \cdot 10^{17}), (\mathrm{Ste2}\#\neg\mathrm{a}, 10^{18}), (\mathrm{G}\beta\gamma{:}\mathrm{G}\alpha\#\mathrm{GDP}, 10^{18})\}$$

$$M_2 = (R_{21}, R_{22}, R_{23}, R_{24}, R_{25}, R_{26}, \mathrm{f}_{21}, \mathrm{f}_{22}, \mathrm{f}_{23}, \mathrm{f}_{24}, \mathrm{f}_{25}, \mathrm{f}_{26}, A_2) \quad \text{with}$$
$$R_{21} = \mathrm{Ste5} + \mathrm{Ste11} \longrightarrow \mathrm{Ste5{:}Ste11}$$
$$R_{22} = \mathrm{Ste5{:}Ste11} \longrightarrow \mathrm{Ste5} + \mathrm{Ste11}$$

$$
\begin{aligned}
R_{23} &= \text{Ste7} \ + \ \text{Fus3} \ \longrightarrow \ \text{Ste7:Fus3} \\
R_{24} &= \text{Ste7:Fus3} \ \longrightarrow \ \text{Ste7} \ + \ \text{Fus3} \\
R_{25} &= \text{Ste5:Ste11} \ + \ \text{Ste7:Fus3} \ \longrightarrow \ \text{Ste5:Ste11:Ste7:Fus3} \\
R_{26} &= \text{Ste5:Ste11:Ste7:Fus3} \ \longrightarrow \ \text{Ste5} \ + \ \text{Ste11} \ + \ \text{Ste7} \ + \ \text{Fus3} \\
\mathrm{f}_{21}(L_2(t)) &= \left\lfloor \mathrm{k}_{21}[\text{Ste5}]_{L_2(t)}[\text{Ste11}]_{L_2(t)}(1/\mathrm{V}_2^2) \right\rfloor \\
\mathrm{f}_{22}(L_2(t)) &= \left\lfloor \mathrm{k}_{22}[\text{Ste5:Ste11}]_{L_2(t)}(1/\mathrm{V}_2) \right\rfloor \\
\mathrm{f}_{23}(L_2(t)) &= \left\lfloor \mathrm{k}_{23}[\text{Ste7}]_{L_2(t)}[\text{Fus3}]_{L_2(t)}(1/\mathrm{V}_2^2) \right\rfloor \\
\mathrm{f}_{24}(L_2(t)) &= \left\lfloor \mathrm{k}_{24}[\text{Ste7:Fus3}]_{L_2(t)}(1/\mathrm{V}_2) \right\rfloor \\
\mathrm{f}_{25}(L_2(t)) &= \left\lfloor \mathrm{k}_{25}[\text{Ste5:Ste11}]_{L_2(t)}[\text{Ste7:Fus3}]_{L_2(t)}(1/\mathrm{V}_2^2) \right\rfloor \\
\mathrm{f}_{26}(L_2(t)) &= \left\lfloor \mathrm{k}_{26}[\text{Ste5:Ste11:Ste7:Fus3}]_{L_2(t)}(1/\mathrm{V}_2) \right\rfloor \\
A_2 &= \{(\text{Ste5}, 9.5 \cdot 10^{16}), (\text{Ste11}, 9.5 \cdot 10^{16}), (\text{Ste7}, 2 \cdot 10^{16}), (\text{Fus3}, 2 \cdot 10^{16})\}
\end{aligned}
$$

$$
\begin{aligned}
M_3 &= (R_{31}, R_{32}, R_{33}, R_{34}, R_{35}, R_{36}, R_{37}, R_{38}, \\
&\qquad \mathrm{f}_{31}, \mathrm{f}_{32}, \mathrm{f}_{33}, \mathrm{f}_{34}, \mathrm{f}_{35}, \mathrm{f}_{36}, \mathrm{f}_{37}, \mathrm{f}_{38}, A_3) \quad \text{with} \\
R_{31} &= \text{Ste5:Ste11:Ste7:Fus3} \ + \ \mathrm{G}\beta\gamma \ \longrightarrow \ \text{Ste5:Ste11:Ste7:Fus3:G}\beta\gamma \\
R_{32} &= \text{Ste5:Ste11:Ste7:Fus3:G}\beta\gamma \ \longrightarrow \ \text{Ste5:Ste11:Ste7:Fus3} \ + \ \mathrm{G}\beta\gamma \\
R_{33} &= \text{Ste5:Ste11:Ste7:Fus3:G}\beta\gamma \ + \ \text{Ste20} \ \longrightarrow \ \mathrm{C}\#\neg\mathrm{p}\#\neg\mathrm{p}\#\neg\mathrm{p}\#\neg\mathrm{p} \\
R_{34} &= \mathrm{C}\#*\#*\#*\#* \ \longrightarrow \ \text{Ste5:Ste11:Ste7:Fus3:G}\beta\gamma \ + \ \text{Ste20} \\
R_{35} &= \mathrm{C}\#\neg\mathrm{p}\#\neg\mathrm{p}\#*\#* \ \longrightarrow \ \mathrm{C}\#\neg\mathrm{p}\#\mathrm{p}\#*\#* \\
R_{36} &= \mathrm{C}\#\neg\mathrm{p}\#\mathrm{p}\#\neg\mathrm{p}\#* \ \longrightarrow \ \mathrm{C}\#\neg\mathrm{p}\#\mathrm{p}\#\mathrm{p}\#* \\
R_{37} &= \mathrm{C}\#\neg\mathrm{p}\#\mathrm{p}\#\mathrm{p}\#\neg\mathrm{p} \ \longrightarrow \ \mathrm{C}\#\neg\mathrm{p}\#\mathrm{p}\#\mathrm{p}\#\mathrm{p} \\
R_{38} &= \mathrm{C}\#\neg\mathrm{p}\#\mathrm{p}\#\mathrm{p}\#\mathrm{p} \ \longrightarrow \ \mathrm{C}\#\mathrm{p}\#\mathrm{p}\#\mathrm{p}\#\mathrm{p} \\
\mathrm{f}_{31}(L_3(t)) &= \left\lfloor \mathrm{k}_{31}[\text{Ste5:Ste11:Ste7:Fus3}]_{L_3(t)}[\mathrm{G}\beta\gamma]_{L_3(t)}(1/\mathrm{V}_3^2) \right\rfloor \\
\mathrm{f}_{32}(L_3(t)) &= \left\lfloor \mathrm{k}_{32}[\text{Ste5:Ste11:Ste7:Fus3:G}\beta\gamma]_{L_3(t)}(1/\mathrm{V}_3) \right\rfloor \\
\mathrm{f}_{33}(L_3(t)) &= \left\lfloor \mathrm{k}_{33}[\text{Ste5:Ste11:Ste7:Fus3:G}\beta\gamma]_{L_3(t)}[\text{Ste20}]_{L_3(t)}(1/\mathrm{V}_3^2) \right\rfloor \\
\mathrm{f}_{34}(L_3(t)) &= \left\lfloor \mathrm{k}_{34}[\mathrm{C}\#*\#*\#*\#*]_{L_3(t)}(1/\mathrm{V}_3) \right\rfloor \\
\mathrm{f}_{35}(L_3(t)) &= \left\lfloor \mathrm{k}_{35}[\mathrm{C}\#\neg\mathrm{p}\#\neg\mathrm{p}\#*\#*]_{L_3(t)}(1/\mathrm{V}_3) \right\rfloor \\
\mathrm{f}_{36}(L_3(t)) &= \left\lfloor \mathrm{k}_{36}[\mathrm{C}\#\neg\mathrm{p}\#\mathrm{p}\#\neg\mathrm{p}\#*]_{L_3(t)}(1/\mathrm{V}_3) \right\rfloor \\
\mathrm{f}_{37}(L_3(t)) &= \left\lfloor \mathrm{k}_{37}[\mathrm{C}\#\neg\mathrm{p}\#\mathrm{p}\#\mathrm{p}\#\neg\mathrm{p}]_{L_3(t)}(1/\mathrm{V}_3) \right\rfloor \\
\mathrm{f}_{38}(L_3(t)) &= \left\lfloor \mathrm{k}_{38}[\mathrm{C}\#\neg\mathrm{p}\#\mathrm{p}\#\mathrm{p}\#\mathrm{p}]_{L_3(t)}(1/\mathrm{V}_3) \right\rfloor \\
A_3 &= \{(\text{Ste20}, 6 \cdot 10^{17})\}
\end{aligned}
$$

$$
\begin{aligned}
M_4 &= (R_{41}, R_{42}, R_{43}, R_{44}, \mathrm{f}_{41}, \mathrm{f}_{42}, \mathrm{f}_{43}, \mathrm{f}_{44}, A_4) \quad \text{with} \\
R_{41} &= \mathrm{C}\#\mathrm{p}\#\mathrm{p}\#\mathrm{p}\#\mathrm{p} \ \longrightarrow \ \mathrm{C}'\#\mathrm{p}\#\mathrm{p}\#\mathrm{p} \ + \ \text{Fus3}\#\mathrm{p} \\
R_{42} &= \mathrm{C}'\#\mathrm{p}\#\mathrm{p}\#\mathrm{p} \ + \ \text{Fus3}\#\neg\mathrm{p} \ \longrightarrow \ \mathrm{C}\#\mathrm{p}\#\mathrm{p}\#\mathrm{p}\#\neg\mathrm{p}
\end{aligned}
$$

$$
\begin{aligned}
R_{43} &= \mathrm{C\#p\#p\#p\#\neg p} \longrightarrow \mathrm{C'\#p\#p\#p} + \mathrm{Fus3\#\neg p} \\
R_{44} &= \mathrm{C\#p\#p\#p\#\neg p} \longrightarrow \mathrm{C\#p\#p\#p\#p} \\
\mathrm{f}_{41}(L_4(t)) &= \left\lfloor \mathrm{k}_{41}[\mathrm{C\#p\#p\#p\#p}]_{L_4(t)}(1/\mathrm{V}_4) \right\rfloor \\
\mathrm{f}_{42}(L_4(t)) &= \left\lfloor \mathrm{k}_{42}[\mathrm{C'\#p\#p\#p}]_{L_4(t)}[\mathrm{Fus3}]_{L_4(t)}(1/\mathrm{V}_4^2) \right\rfloor \\
\mathrm{f}_{43}(L_4(t)) &= \left\lfloor \mathrm{k}_{43}[\mathrm{C\#p\#p\#p\#\neg p}]_{L_4(t)}(1/\mathrm{V}_4) \right\rfloor \\
\mathrm{f}_{44}(L_4(t)) &= \left\lfloor \mathrm{k}_{44}[\mathrm{C\#p\#p\#p\#\neg p}]_{L_4(t)}(1/\mathrm{V}_4) \right\rfloor \\
A_4 &= \{(\mathrm{Fus3}, 6 \cdot 10^{17})\}
\end{aligned}
$$

While most parts of the model given in [6] were directly adapted, a few slight changes had to be introduced. Concentrations, which were given in nM in the original, were converted into molecule numbers. In order to compensate for this effect on the reaction kinetics, these were extended by the volume V_i of each module as a normalization constant. Values for all parameters k_{ij} appearing in the reaction kinetics are not given here, but can be calculated from the data given in [6]. In accordance with the original model, reaction kinetics consist of mass-action formulations, which have to be rounded in order to yield integer values as results.

6 Discussion

An important aspect of the system is its capability to deal with the combinatorial complexity arising when proteins incorporating multiple sites for modification and binding are involved. The usage of wild-cards and exclusions allows a compact formulation of systems that would be substantially larger if only atomic objects were considered. This major advantage could even be extended by the introduction of generic logical expressions into the protein descriptions.

It is important to mention two potential limitations of the system. On the one hand, the reaction kinetics mechanism might produce imprecise results in case of small object numbers, due to the fact that educts of multiple reactions can sum up to more proteins than currently present. As a possible solution, we propose the extension of the multiset definition to negative multiplicities, so that modules can formally contain negative numbers of objects. In this case, the kinetic formulation would ensure any educt with a negative multiplicity could not participate in a reaction, but rather would have to be refilled again. On the other hand, a potential constraint comes from the way in which the kinetic functions are applied to each combination of proteins matching the right side of a reaction rule. In order to work precisely, this approach requires the kinetic functions to be linear. Therefore, it is advisable not to use wild-carded reaction rules with non-linear kinetics. Both of these points require further research to maximize the system's usability.

The choice of the matching strategy strongly influences the system behavior. Strict matching implies maximum specificity of the reactions. In contrast, to

involve a large pool of protein configurations, a loose matching should be preferred. Special application scenarios can be tackled by additional matching strategies.

7 Conclusions

The introduced model Π_{CSN} intends to combine advantages of P systems with mechanisms observed in cell signalling networks. It is conceived as a description, analysis, and prediction tool for ongoing studies about the evolutionary development of protein structures, their properties, interactions, and resulting network functions. To this end, we have integrated string-objects and a modular architecture into a deterministic framework. In future, simulations as well as theoretical investigations will lead the way towards a deeper understanding of the correlation between CSN structure and function.

Acknowledgements

This work is part of the ESIGNET project (Evolving Cell Signalling Networks in Silico), which has received research funding from the European Community's Sixth Framework Programme (project no. 12789). Further funding from the Federal Ministry of Education and Research (BMBF, grant 0312704A) is acknowledged.

References

1. L. Bardwell. A walk-through of the yeast mating pheromone response pathway. *Peptides*, 25:1465–1476, 2004.
2. F. Bernardini, M. Gheorghe. Population P systems. *Journal of Universal Computer Science*, 10(5):509–539, 2004.
3. L. Cardelli, A.D. Gordon. Mobile ambients. LNCS 1378, Springer-Verlag, Berlin, 1998.
4. S. Eilenberg. *Automata, Languages, and Machines.* Academic Press, New York, 1976.
5. D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81:2340–2361, 1977.
6. B. Kofahl, E. Klipp. Modelling the dynamics of the yeast pheromone pathway. *Yeast*, 21:831–850, 2004.
7. M.O. Magnasco. Chemical kinetics is Turing universal. *Physical Review Letters*, 78(6):1190–1193, 1997.
8. V. Manca, L. Bianco, F. Fontana. Evolution and oscillation in P systems: Applications to biological phenomena. LNCS 3365, Springer-Verlag, Berlin, 2005.
9. C. Martin-Vide, G. Păun. Computing with membranes (P systems): Universality results. LNCS 2055, Springer-Verlag, Berlin, 2001.
10. C. Martin-Vide, G. Păun, J. Pazos, A. Rodriguez-Paton. Tissue P systems. *Theoretical Computer Science*, 296(2):295–326, 2003.

11. R. Milner. *Communicating and Mobile Systems: the Pi-Calculus.* Cambridge University Press, 1999.
12. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
13. G. Păun. *Membrane Computing: An Introduction.* Springer-Verlag, Berlin, 2002.
14. G. Păun, Y. Sakakibara, T. Yokomori. P systems on graphs of restricted forms. *Publicationes Mathematicae*, 60, 2002.
15. D. Pescini, D. Besozzi, G. Mauri, C. Zandron. Dynamical probabilistic P systems. *International Journal of Foundations of Computer Science*, 17(1):183–195, 2006.
16. J.L. Peterson. *Petri Net Theory and the Modelling of Systems.* Prentice Hall, 1961.
17. G. Rozenberg, A. Salomaa, Eds. *Handbook of Formal Languages.* Springer-Verlag, Berlin, 1999.
18. Y. Suzuki, H. Tanaka. Symbolic chemical system based on abstract rewriting and its behavior pattern. *Artificial Life and Robotics*, 1:211–219, 1997.

Characterizations of Some Restricted Spiking Neural P Systems*

Oscar H. Ibarra and Sara Woodworth

Department of Computer Science
University of California, Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu

Abstract. A k-output spiking neural P system (SNP) with output neurons, $O_1, \cdots, O_k$, generates a tuple $(n_1, \cdots, n_k)$ of positive integers if, starting from the initial configuration, there is a sequence of steps such that during the computation, each O_i generates exactly two spikes a a (the times the pair a a are generated may be different for different output neurons) and the time interval between the first a and the second a is n_i. After the output neurons have generated their pairs of spikes, the system eventually halts. Another model, called k-train SNP, has only one output neuron. It generates a k-tuple $(n_1, \cdots, n_k)$ if, starting from the initial configuration, the output neuron O generates the spike train $aa \cdots a$ with exactly $k+1$ a's such that the interval between the i^{th} a and the $i+1^{st}$ a is n_i, and the system eventually halts. We assume, without loss of generality, that each neuron in the SNP is either bounded or unbounded. (Bounded here means that there is a fixed constant c such that at any time during the computation, the number of spikes in the neuron is at most c. Otherwise, the neuron is unbounded.) It is known that 1-output SNPs (= 1-train SNPs) are universal, i.e., they generate exactly the recursively enumerable sets over N. Here, we show the following:

1. For $k \geq 1$, a set $Q \subseteq N^k$ is semilinear if and only if it can be generated by a k-output SNP, where every unbounded neuron satisfies the property that once it starts "spiking" it will no longer receive future spikes (but can continue spiking). This result also holds for k-train SNP.
2. The set $Q = \{(m, 2m) \mid m \geq 1\}$ (which is semilinear) cannot be generated by any 2-output bounded SNP (i.e., SNP all of whose neurons are bounded). Thus, for $k \geq 2$, there are semilinear sets over N^k that cannot be generated by k-output bounded SNPs. This contrasts a known result that 1-output bounded SNPs generate all semilinear sets over N.
3. For $k \geq 2$, k-output bounded SNPs are computationally more powerful than k-train bounded SNPs. (They are identical when $k = 1$.)
4. For $k \geq 1$, k-output bounded SNPs and k-train bounded SNPs can be characterized by certain classes of nondeterministic finite automata with strictly monotonic counters.

* This research was supported in part by NSF Grants CCF-0430945 and CCF-0524136.

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 424–442, 2006.

1 Introduction

Spiking neural P systems (SNPs) were recently introduced in [6], and investigated in a series of papers: [11], [12], [5]. The SNP model incorporates into membrane computing [10] ideas from spiking neurons, see, e.g., [1], [7], [8].

A SNP consists of a set of neurons corresponding to nodes of a graph. The neurons send signals (spikes) along synapses (directed edges of the graph). This is done by means of firing rules, which are of the form $E/a^j \to a^k; t$ with $j \geq 1, j \geq k \geq 0, t \geq 0$, where E is a regular expression, j is the number of spikes consumed by the rule, k is the number of spikes transmitted along each outgoing synapse, and t is the delay from firing the rule and emitting the spike. The rule can be used only if the number of spikes in the neuron is "covered" by expression E, in the sense that the current number of spikes in the neuron, n, is such that a^n is contained in the set $L(E)$ denoted by the expression E. In the time interval between firing a rule and emitting the spike, the neuron is closed/blocked – it does not receive other spikes and cannot fire. After the time interval, the neuron is again open and can again fire and receive other spikes. As a simplification, the rule $a^j/a^j \to a; t$ is written as $a^j \to a; t$. Here as in previous papers we will only use rules where $k = 0$ or 1. When $k = 0$, we call this rule a forgetting rule. We only use this type of rule in a very restricted sense. For the forgetting rules in this paper, the delay is always 0, E is a singleton language, all spikes in E are used when the rule is used, and E is disjoint from all the other rules in the neuron. We use the rule form $a^j \to \lambda$ to represent these rules. (This definition is chosen since it is consistent with the definition in all previous papers.)

Starting from a fixed initial distribution of spikes in the neurons (initial configuration) and using the rules in a synchronized manner (a global clock is assumed), the system evolves. A computation is a sequence of transitions starting from the initial configuration. A transition is maximally parallel in the sense that all neurons that are fireable must fire. However, in any neuron, at most one rule is allowed to fire. Details can be found in [6].

SNPs can be used as computing devices in various ways. Here, as in previous papers, we will use them as generators of numbers. We will only consider SNPs with two types of neurons:

1. A neuron is *bounded* if every rule in the neuron is of the form $a^i/a^j \to a; t$, where $j \leq i$, or of the form $a^k \to \lambda$, provided there is no rule of the form $a^k/a^j \to a; t$ in the neuron. Note that there can be several such rules in the neuron. These rules are called *bounded rules*. (For notational convenience, we will write $a^i/a^i \to a; t$ simply as $a^i \to a; t$.)
2. A neuron is *unbounded* if every rule in the neuron is of the form $E/a^c \to a; t$, where E denotes an infinite (unary) regular set. Examples of such rules are: $a^3(a^2)^*/a^2 \to a; t$, $a^3(a^2)^*/a^3 \to a; t$, $a(a)^*/a \to a; t$. (Again, there can be several such rules in the neuron.) These rules are called *unbounded rules*.

A SNP is bounded if all the neurons in the system are bounded. If, in addition, there are unbounded neurons then the SNP is said to be unbounded.

We generalize the SNP by allowing it to produce k outputs. A *k-output SNP* Π has k output neurons, $O_1, \cdots, O_k$. We say that Π generates a k-tuple $(n_1, \cdots, n_k) \in N^k$ if, starting from the initial configuration, there is a sequence of steps such that each output neuron O_i generates exactly two spikes a a (the times the pairs a a are generated may be different for different output neurons) and the time interval between the first a and the second a is n_i.

Moreover, after all the output neurons have generated their pair of spikes, the system eventually *halts* in the sense that it reaches a configuration where all neurons are open but no neurons are fireable. The set of all k-tuples generated is denoted by $Q(\Pi)$.

It was recently shown in [5] that a set $Q(\Pi) \subseteq N^1$ is recursively enumerable if and only if it can be generated by a 1-output unbounded SNP Π all of whose unbounded neurons have only one rule, and it is either $a(a)^*/a \rightarrow a; 0$ or $a(a)^*/a \rightarrow a; 2$.

It was also shown in [6] that semilinear subsets of N^1 can be characterized by bounded SNPs. However, it turns out that bounded k-output SNPs cannot generate all semilinear subsets of N^k, when $k \geq 2$. For example, we show in this paper that the set of tuples $\{(m, 2m) \mid m \geq 1\}$ cannot be generated by a 2-output SNP. We then give a characterization of subsets of N^k generated by k-output bounded SNPs.

For semilinear subsets of N^k (for any k), we show that they can be characterized by k-output unbounded SNPs where every unbounded neuron satisfies the property that that once it starts "spiking" it will no longer receive future spikes (but can continue spiking). Such SNPs are called 1-reversal-bounded.

We then look at another restricted model called *k-train SNP*. Such a system has only one output neuron. Again, there are two types of neurons: bounded and unbounded but 1-reversal-bounded, as defined before. We say that Π generates the k-tuple $(n_1, \cdots, n_k)$ if, starting from the initial configuration, the output neuron O generates the spike train $aa \cdots a$ with exactly $k+1$ outputted a's such that the interval between the i^{th} a and the $i+1^{st}$ a is n_i, and the system eventually halts. We show that 1 reversal-bounded k-train unbounded SNPs also characterize the semilinear sets over N^k. We also give a characterization of k-train bounded SNPs and show that they cannot generate all semilinear sets. In fact, they are weaker than k-output bounded SNPs.

2 Characterization of k-Output Bounded SNPs

We first recall the definition of a semilinear set. A set $Q \subseteq N^k$ is a *linear set* if there exist vectors $v_0, v_1, \ldots, v_t$ in N^k such that

$$Q = \{v \mid v = v_0 + m_1 v_1 + \cdots + m_t v_t,\ m_i \in N\}.$$

The vectors v_0 (referred to as the *constant vector*) and $v_1, v_2, \ldots, v_t$ (referred to as the *periods*) are called the *generators* of the linear set Q. A set $Q \subseteq N^k$ is *semilinear* if it is a finite union of linear sets. The empty set is a trivial (semi)linear set, where the set of generators is empty. Every finite subset of N^k is semilinear – it

is a finite union of linear sets whose generators are constant vectors. It is also clear that the semilinear sets are closed under (finite) union. It is also known that they are closed under complementation, intersection, and projection. A semilinear subset of N^1 (i.e., 1-tuples) is sometimes referred to as regular.

It is known that 1-output SNPs with only bounded neurons generate exactly the semilinear sets over N^1 [6]. However, as we shall see in Theorem 1, when $k \geq 2$, k-output SNPs with only bounded neurons cannot generate all semilinear sets over N^k. But these SNPs can generate some simple semilinear sets. For example Figure 1 is a SNP with only bounded neurons generating the set $\{(m, n, m+n) \mid m, n \geq 2\}$.

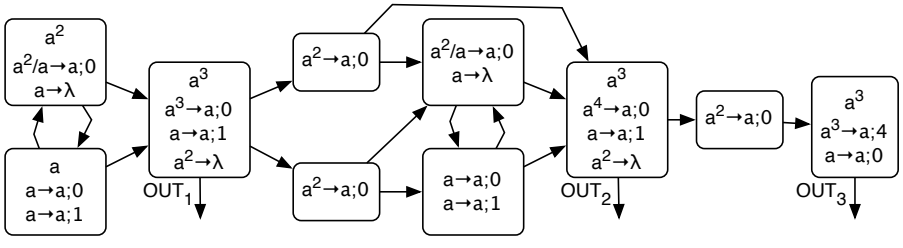


Fig. 1. Bounded SNP generating $Q = \{(m, n, m+n) \mid m, n \geq 2\}$

Theorem 1. *The set $Q = \{(m, 2m) \mid m \geq 1\}$ cannot be generated by any 2-output bounded SNP.*

Proof. Let Π be a SNP with m neurons that are all bounded. The distribution of the spikes in the neurons and states of the neurons corresponding to the spiking intervals specified by the last rules used in each neuron (the open-close status and the time since the neurons were closed, depending on the rule used) and the rule to be used next define the configuration of the system. It is important to note that our definition of configuration includes the next rule that can be applied in the neuron. Clearly, there are at most n configurations that the system can be in, for some n. Let O_1 and O_2 be the output neurons.

Suppose Π generates the set $Q = \{(m, 2m) \mid m \geq 1\}$. Let $s = n + 1$. Then Π generates $(s, 2s)$. Let t_1 and t_2 be the times the output membrane O_1 generates the first and second spikes, respectively. Hence, $t_2 - t_1 = s$. Similarly, let t_3 and t_4 be the times O_2 spikes; hence $t_4 - t_3 = 2s$. We consider two cases:

Case 1. Suppose $t_1 \leq t_3$. Then we claim that $t_3 - t_1 \leq n$. Otherwise, in the time interval between t_1 and t_3, the system will enter some configuration C twice at times t'_1 and t'_3, where $t_1 \leq t'_1 < t'_3 \leq t_3$. Let $t'_3 - t'_1 = r$. (Note that $r \leq n$.) Then, clearly the tuple $(s + kr, 2s)$ is also generated by Π for each $k \geq 1$. This is a contradiction.

Case 2. Now suppose $t_3 \leq t_1$. Then, again, $t_1 - t_3 \leq n$; otherwise (by an argument similar to Case 1), for some $r \leq n$, $(s, 2s + kr)$ will also be generated by Π for each $k \geq 1$, a contradiction.

Thus we may assume that $|t_3 - t_1| \leq n$. But this would imply that $t_4 - t_2 \geq n$. Then for some $r \leq n$, $(s, 2s + kr)$ will also be generated by Π for each $k \geq 1$, which is again a contradiction. □

Next, we give a characterization of subsets of N^k generated by k-output bounded SNPs. Consider a nondeterministic finite automaton $\mathcal{M}$ with k counters, all of which are output counters.

1. Initially all counters are zero.
2. No counter is decremented during the computation.
3. In one step, zero or more counters can be incremented (by 1).
4. When a counter is incremented, it gets incremented at every step until such time when it stops incrementing. Thereafter, the counter no longer increments.
5. Each counter gets incremented at some point.
6. When all the counters have stopped incrementing, the machine may continue computing but eventually halts in an accepting state. The values in the k counters is then said to be generated by $\mathcal{M}$.

Note that item 4 is an important restriction on the operation of the machine. Call the counter machine (CM) described above a *strictly monotonic k-output CM*. Clearly, the set $\{(m, n, m + n) \mid m, n \geq 1\}$ can be generated by a strictly monotonic 3-output CM. Other examples are: $\{(n, n) \mid n \geq 1\}$, $\{(k, n) \mid k, n \geq 1, k < n\}$, and $\{(m, k, n,) \mid m, k, n \geq 1, m < k < n\}$. The first two can be generated by strictly monotonic 2-output CMs, and the last can be generated by a strictly monotonic 3-output CM.

Formally, we can define a strictly monotonic k-output CM as $\mathcal{M} =< k, B, l_1, l_h, R >$ where k is the number of counters in the system, B is the set of instruction labels, l_1 is the starting instruction, l_h is the halting instruction, and R is the set of instructions. The instructions in R are of the form

$l_i : (\texttt{BEGIN}(r_1, \cdots, r_p), \texttt{END}(s_1, \cdots, s_q), l_{i_1}, \cdots, l_{i_t})$
$l_i : (\texttt{CONTINUE}, l_{i_1}, \cdots, l_{i_t})$
$l_i : (\texttt{HALT})$

The counters are indexed $1, \cdots, k$. The instruction $l_i : (\texttt{BEGIN}(r_1, \cdots, r_p), \texttt{END}(s_1, \cdots, s_q), l_{i_1}, \cdots, l_{i_t})$ starts incrementing counters $r_1, \cdots, r_p$ (this assumes these counters were zero) and stops incrementing counters $s_1, \cdots, s_q$ (this assumes these counters have been incrementing at every step, since they started incrementing earlier). The set $\{r_1, \cdots, r_p\}$ (respectively, $\{s_1, \cdots, s_q\}$) is called the incrementing set (respectively, end-incrementing set) of the instruction. The counters in the incrementing set begin incrementing during the current step, but the counters in the end-incrementing step are stopped before they are incremented during the current step. If $p = 0$ or $q = 0$, we do not write BEGIN or END. In particular, if $p = q = 0$, then the instruction reduces to $l_i : (l_{i_1}, \cdots, l_{i_t})$, which we denote by the instruction $l_i : (\texttt{CONTINUE}, l_{i_1}, \cdots, l_{i_t})$ for clarity. During this instruction no counter changes state but all previously incrementing counters are incremented by one and then the system changes state. We emphasize that

once a counter r has begun incrementing, it is incremented at each proceeding step until r is in the end-incrementing set of the current instruction. If r is ever incremented again, the computation is invalid. The incrementing set must be disjoint of the end-incrementing set (since we cannot begin and end a counter in a single step). The instruction $l_i : (\texttt{HALT})$ halts the execution. We can assume without loss of generality that no instruction loops back to itself. Note that if $l_{i_1}, \cdots, l_{i_t}$ are identical (i.e., the next instruction is unique), we need only put one such label.

To better understand how strictly monotonic CMs operate we give the following examples.

Example 1. The set $Q = \{(n, n) \mid n \geq 1\}$ can be generated by the strictly monotonic CM $\mathcal{M} =< 2, \{l_1, \cdots, l_5\}, l_1, l_5, R >$ where

$$\begin{aligned} R = \{\ & l_1 : (\texttt{BEGIN}(1, 2), l_2, l_4), \\ & l_2 : (\texttt{CONTINUE}, l_3, l_4), \\ & l_3 : (\texttt{CONTINUE}, l_2, l_4), \\ & l_4 : (\texttt{END}(1, 2), l_5), \\ & l_5 : (\texttt{HALT})\} \end{aligned}$$

This program begins incrementing the counters at time 1. The (possibly executed) `CONTINUE` instructions allow some nondeterministic amount of time to elapse. Then we end the incrementing of both counters at time $n + 1$. This gives us the tuple (n, n) where $n \geq 1$. (Note that the counters increment during the time step they are begun guaranteeing that each counter contains an output which is ≥ 1.)

Example 2. The set $Q = \{(m, k, n) \mid m \leq k \leq n, m \geq 1\}$ can be generated by the strictly monotonic CM $\mathcal{M} =< 3, \{l_1, \cdots l_{14}\}, l_1, l_{14}, R >$ where

$$\begin{aligned} R = \{\ & l_1 : (\texttt{BEGIN}(1, 2, 3), l_2, l_4, l_5, l_6), \\ & l_2 : (\texttt{CONTINUE}, l_3, l_4.l_5, l_6), \\ & l_3 : (\texttt{CONTINUE}, l_2, l_4, l_5, l_6), \\ & l_4 : (\texttt{END}(1), l_7, l_9, l_{10}), \\ & l_5 : (\texttt{END}(1, 2), l_{11}, l_{13}), \\ & l_6 : (\texttt{END}(1, 2, 3), l_{14}), \\ & l_7 : (\texttt{CONTINUE}, l_8, l_9, l_{10}), \\ & l_8 : (\texttt{CONTINUE}, l_7, l_9, l_{10}), \\ & l_9 : (\texttt{END}(2), l_{11}, l_{13}), \\ & l_{10} : (\texttt{END}(2, 3), l_{14}), \\ & l_{11} : (\texttt{CONTINUE}, l_{12}, l_{13}), \\ & l_{12} : (\texttt{CONTINUE}, l_{11}, l_{13}), \\ & l_{13} : (\texttt{END}(3), l_{14}), \\ & l_{14} : (\texttt{HALT})\} \end{aligned}$$

The following characterizes k-output bounded SNPs.

Theorem 2. *A set $Q \subseteq N^k$ is generated by a k-output bounded SNP if and only if Q is generated by a strictly monotonic k-output CM.*

Proof. Let Π be a bounded SNP with output neurons $O_1, \cdots, O_k$. Denote by $c_1, \cdots, c_n$ all possible configurations Π can be in, with c_1 the initial configuration and c_n the halting configuration. The configuration contains the number of spikes in each neuron, the open/closed status of each neuron, the current delay for each closed neuron, and the number of times each output neuron has spiked so far (0, 1, 2, or more than 2 times). The strictly monotonic CM $\mathcal{M}$ simulating Π will have counters $1, \ldots, k$, where counter i corresponds to output neuron O_i. The instructions of $\mathcal{M}$ are defined as follows:

1. c_n : (HALT) is an instruction of $\mathcal{M}$.
2. If from configuration c_i, Π enters configuration c_{i_1} or c_{i_2} or $\cdots$ or c_{i_t} without any of the output neurons spiking, then $c_i : (\texttt{CONTINUE}, c_{i_1}, c_{i_2}, \cdots, c_{i_t})$ is an instruction of $\mathcal{M}$.
3. If from configuration c_i, Π enters configuration c_{i_1} or c_{i_2} or $\cdots$ or c_{i_t} with output neurons $O_{r_1}, \cdots, O_{r_p}$ spiking for the first time and output neurons $O_{s_1}, \cdots, O_{s_q}$ spiking for the second time, then $l_i : (\texttt{BEGIN}(r_1, \cdots, r_p), \texttt{END}(s_1, \cdots, s_q), l_{i_1}, \cdots, l_{i_t})$ is an instruction of $\mathcal{M}$. (If some output neuron spikes more than twice, we force $\mathcal{M}$ into an infinite loop.)

It is straightforward to verify that $\mathcal{M}$ generates exactly the tuples generated by Π.

We now prove the converse. Let $\mathcal{M}$ be a strictly monotonic k-output CM. We construct a k-output bounded SNP Π equivalent to $\mathcal{M}$. Informally, Π will simulate each instruction of $\mathcal{M}$. When a instruction begins incrementing counter r, the corresponding output neuron is triggered to spike sending the first a into the environment. When a instruction is executed which ends the incrementing of counter r the corresponding output neuron is triggered to spike sending a second a into the environment. (If a invalid computation is simulated, some output neuron will end up sending at least 3 spikes into the environment invalidating the simulation.) The simulation of each instruction takes *exactly* one time step. This guarantees that the number of time steps between the two spikes of the output neuron is precisely the same as the number of increments of the simulated counter.

Formally, the simulation is done by creating $2m$ modules for each instruction l_i where m is the largest number of nondeterministic choices per instruction in $\mathcal{M}$. (Each module is known as module l_{ij} or l'_{ij} where $1 \leq j \leq m$.) All the neurons are empty in the initial configuration except the neurons in module l_{11} which each initially contain one spike. When the computation halts, every neuron in the system is empty.

To simulate an instruction of the form $l_i : (\texttt{BEGIN}(r_1, \cdots, r_p), \texttt{END}(s_1, \cdots, s_q), l_{i_1}, \cdots, l_{i_t})$ we create $2m$ modules. The first module (l_{i1}) is shown in Figure 2. The module introduces up to $t+2$ new neurons which are initiated when only a single spike is sent to each neuron. At any time step, up to $2m+1$ spikes could be received by any neuron but if more than one spike is received by the neurons in this module they are forgotten (using the rules $a^2 \to \lambda, \cdots, a^{2m+1} \to \lambda$).

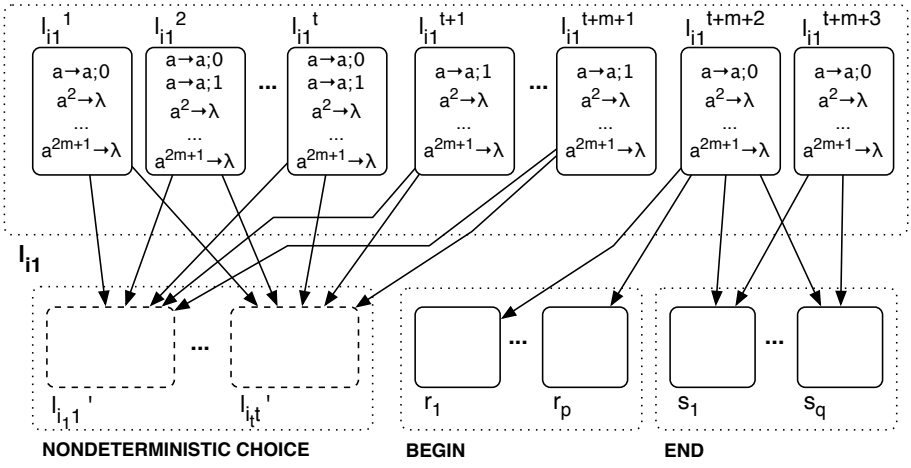


Fig. 2. Bounded SNP module simulating instruction l_{i1} : (BEGIN$(r_1, \cdots, r_p)$, END$(s_1, \cdots, s_q), l_{i_1}, \cdots, l_{i_t}$)

The neuron l_{i1}^{t+m+2} is used to send a single spike to each output neuron in the set $\{r_1, \cdots, r_p, s_1, \cdots, s_q\}$. The neuron l_{i1}^{t+m+3} is used to send a single spike to each output neuron in the set $\{s_1, \cdots, s_q\}$. These two neurons guarantee that when this instruction is simulated, neurons $r_1, \cdots, r_p$ each receive one spike while neurons $s_1, \cdots, s_q$ each receive two spikes. This will cause each of these output neurons to spike in the following step. The different number of spikes allow the output neurons to check that they receive exactly one BEGIN instruction followed by exactly one END instruction. Otherwise the output neuron is guaranteed to spike zero, one, or at least three times.

The set of neurons $l_{i1}^1, \cdots, l_{i1}^{t+m+1}$ is used to nondeterministically choose which instruction to execute next. The neuron l_{i1}^1 will always fire immediately. The neurons $l_{i1}^{t+1}, \cdots l_{i1}^{t+m+1}$ will always fire with a delay of one time step. The remaining neurons $(l_{i1}^2, \cdots, l_{i1}^t)$ nondeterministically choose to either fire immediately or after a delay of one time step. This will produce two sets of spikes. The number of spikes which are sent immediately is i' where $1 \leq i' \leq t$. This set of spikes will initiate exactly one instruction module (module $l'_{i_{i'}i'}$ corresponding to instruction $l_{i'}$) while the other connected modules forget their spikes. The second set of spikes is sent after a delay of one time step. This set consists of $t + m + 1 - i'$ spikes. This set of spikes is forgotten by all the neurons since it is strictly larger than m. (If l_i is a deterministic rule, the neurons $l_{i1}^2, \cdots, l_{i1}^{m+2}$ are unnecessary since their job is to guarantee that the delayed spikes are always forgotten. In a deterministic rule there will be no delayed spikes.)

The instruction modules alternately execute primed and unprimed next instruction modules. So an unprimed module connects only to primed modules and vice versa. This guarantees that a module is unable to get both a set of first

spikes from one instruction module during the same step as the second set of spikes from a different instruction module.

The remaining modules for instruction l_i are created identically to module l_{i1} except that the rules in the neurons are slightly changed. For modules l_{ij} and l'_{ij} (where $1 < j \leq m$), each neuron has spiking rules of the form $a^j \to a; t$ along with the set of forgetting rules $\{a^1 \to \lambda, \cdots, a^{j-1} \to \lambda, a^{j+1} \to \lambda, a^{2m+1} \to \lambda\}$. Depending on the program of $\mathcal{M}$, some modules may be unreachable in the overall simulation. Obviously if the neurons are unreachable they can be removed without detriment.

To simulate the instruction l_i : (CONTINUE, $l_{i_1}, \cdots, l_{i_t}$) we again create $2m$ modules using the same technique. However, since these instructions do not change the state of any counter, the neurons l_{ij}^{t+m+2} and l_{ij}^{t+m+3} (for $1 \leq j \leq m$) are not needed.

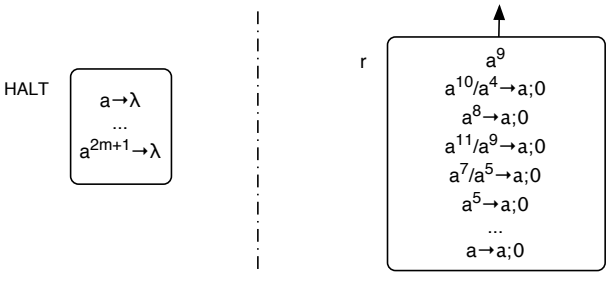


Fig. 3. Bounded SNP module simulating instruction l_i : (HALT) (left) and rules within each output neuron (right)

The halt instruction must simply stop the computation so the simulation module (shown in Figure 3) takes any spikes received and forgets them. Since no further module is initiated, this will cause Π to halt. Also, the rules contained in each output neuron are given in Figure 3. These rules guarantee the neuron receives exactly one BEGIN instruction followed by exactly one END instruction during the computation. Any deviation will guarantee that the valid condition of 'exactly 2 spikes are sent to the environment' is not met.

The SNP system Π that is created outputs exactly two spikes from each output neuron if and only if the simulated computation of $\mathcal{M}$ is valid. □

To illustrate the construction of the bounded SNP simulating a strictly monotonic CM, we give a very simple example below.

Example 3. Let $Q = \{(n, n) \mid n \geq 4, n \text{ is even}\}$. A CM generating this set is $\mathcal{M} =< 2, (l_1, \cdots, l_5), l_1, l_5, R >$ where

$$
\begin{aligned}
R = \{\ & l_1 = (\texttt{BEGIN}(1,2), l_2),\\
& l_2 : (\texttt{CONTINUE}, l_3),\\
& l_3 : (\texttt{CONTINUE}, l_2, l_4),\\
& l_4 : (\texttt{END}(1,2), l_5),\\
& l_5 : (\texttt{HALT})\}
\end{aligned}
$$

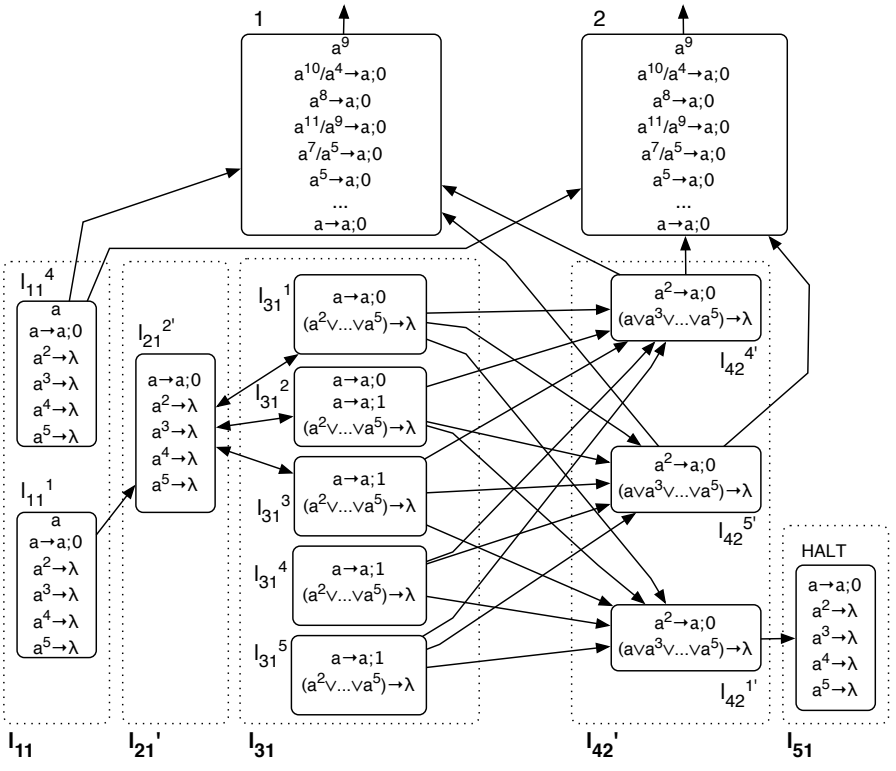


Fig. 4. Bounded SNP Π simulating $\mathcal{M}$ generating $Q = \{(n, n) \mid n \geq 4, n \text{ is even}\}$

The SNP Π which simulates $\mathcal{M}$ is given in Figure 4. (To save space, a set of forgetting rules is written as a rule of type $(a^2 \vee \cdots \vee a^5) \to \lambda$. This means if the neuron contains between 2 and 5 spikes, these spikes are all forgotten.) Note that all unnecessary and unreachable modules and neurons have been removed. It can be seen that for this example, only one module is needed for each instruction. We use module l_{11} to simulate instruction l_1, module l_{21} to simulate instruction l_2, module l_{31} to simulate instruction l_3, and module l_{42} to simulate instruction l_4. Also, many of these modules had unnecessary neurons which were also removed.

The characterization allows us to easily show that a set is generated by a k-output bounded SNP by simply showing that it can be generated by a strictly monotonic k-output CM. So, e.g., the sets in Examples 1 and 2 can be generated by 2-output and 3-output bounded SNPs, respectively.

Suppose we relax the operation of a strictly monotonic k-output CM so that we no longer require that when a counter is incremented, it gets incremented at every step until such time when it stops incrementing. Thus, each counter need not be incremented at each step, but eventually, the machine halts in an accepting state when each counter has value at least 1. This type of machine (called

0-reversal CM in the next section) is more powerful than a strictly monotonic k-output CM, since such CMs generate exactly the semilinear sets. In the next section, we characterize them in terms of restricted k-output unbounded SNPs.

3 Reversal-Bounded k-Output Unbounded SNPs

In order to characterize all the semilinear sets over N^k, we need unbounded neurons that operate in a restricted manner. A k-output SNP is *reversal-bounded* if there is an integer $r \geq 0$ such that for each unbounded neuron, the number of times the spike size changes values from nonincreasing to nondecreasing and vice-versa during any computation is at most r. The system is 1-reversal when $r = 1$. A k-output SNP where each unbounded neuron operates with the property that it can receive spikes, but once it starts spiking it will no longer receive future spikes (but can continue spiking) would be considered a 1-reversal SNP. Moreover, as we shall see, for the results of this section, we can assume there is only one rule in each unbounded neuron, and it is $a^3(a^2)^*/a^2 \rightarrow a; 0$. Note that when $r = 0$, i.e., the system is 0-reversal, then the unbounded neurons can be deleted from the system without affecting the computation. Hence, such a system is equivalent to a k-output bounded SNP.

We need a counter machine characterization of semilinear sets. A nondeterministic multicounter machine (CM) $\mathcal{M}$ is a nondeterministic finite automaton with a finite number of counters (it has no input tape). Each counter can only hold a nonnegative integer. The machine starts in a fixed initial state with all counters zero. During the computation, each counter can be incremented by 1, decremented by 1, or tested for zero. A distinguished set of k counters (for some $k \geq 1$) is designated as the output counters. The output counters are non-decreasing (i.e., cannot be decremented). A k-tuple $(n_1, \cdots, n_k) \in N^k$ is generated if $\mathcal{M}$ eventually halts in an accepting state, all non-output counters zero, and the contents of the output counters are $n_1, \cdots, n_k$, respectively. We will refer to a CM with k output counters (the other counters are auxiliary counters) as a k-output CM.

A CM is reversal-bounded if there exists an $r \geq 0$ such that during any computation, each non-output counter is r-reversal in the sense that it alternates between nondecreasing mode and nonincreasing mode and vice-versa at most r times. Note that when $r = 0$, the counter is nondecreasing. (By definition, the output counters are 0-reversal.)

The following theorem is known [4] (see also [3]):

Theorem 3. *The following statements are equivalent for any set $Q \subseteq N^k$:*

1. *Q is a semilinear set.*
2. *Q can be generated by a reversal-bounded k-output CM.*
3. *Q can be generated by a CM with exactly k counters, all of which are output counters (hence, 0-reversal counters).*

Using the above result, we can prove:

Theorem 4. *The following statements are equivalent for any $Q \subseteq N^k$:*

1. *Q is semilinear.*
2. *Q can be generated by a reversal-bounded k-output unbounded SNP.*
3. *Q can be generated by a 1-reversal k-output unbounded SNP.*

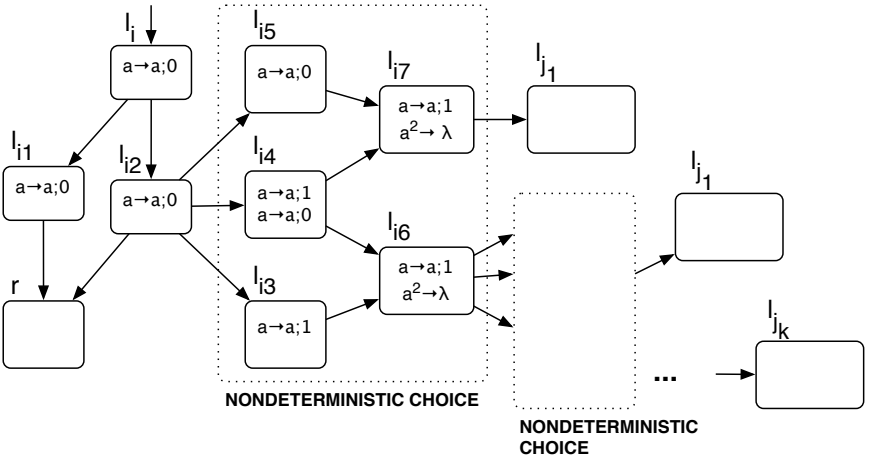


Fig. 5. Reversal-bounded addition module

Proof. Clearly (3) implies (2). We first show that (1) implies (3) and then show that (2) implies (1). From Theorem 3, if Q is semilinear, then it can be generated by a CM $\mathcal{M}$ with exactly k counters, all of which are output counters (hence, nondecreasing). We give a construction of a 1-reversal k-output SNP Π equivalent to $\mathcal{M}$. We will see that each bounded neuron has a spike bound of 2 and each unbounded neuron is 1-reversal. A nondecreasing counter machine has instructions of the form $l_i : (\texttt{ADD}(r), l_{j_1}, \cdots, l_{j_k})$ and $l_i : (\texttt{HALT})$. The instruction $l_i : (\texttt{HALT})$ halts the computation. The instruction $l_i : (\texttt{ADD}(r), l_{j_1}, \cdots, l_{j_k})$ adds 1 to counter r and then nondeterministically moves to some state $l_{j_1} \cdots, l_{j_k}$. To simulate each instruction of the form $l_i : (\texttt{ADD}(r), l_{j_1}, \cdots, l_{j_k})$, we create an addition module as shown in Figure 5. To simulate the instruction of the form $l_i : (\texttt{HALT})$ we create the output module shown in Figure 6 which halts the computation and outputs our generated k-tuple. The initial configuration of Π has zero spikes in all neurons except the neuron l_0 which contains a single spike.

Each addition module is initiated when a spike enters neuron l_i. This causes neuron l_i to spike transmitting a spike to both l_{i1} and l_{i2}. Both of these neurons in turn spike sending 2 spikes to neuron r. Neuron r records the current count of counter r (x_r) by containing $2x_r$ spikes. Neuron l_{i2} also sends three spikes to neurons l_{i3}, l_{i4}, and l_{i5} (which operate with neurons l_{i6} and l_{i7} to form a nondeterministic module). These three neurons use rules with various delays to nondeterministically either have neuron l_{i6} or neuron l_{i7} fire (the other neuron forgets). This initiates either the simulation of instruction l_j or triggers another

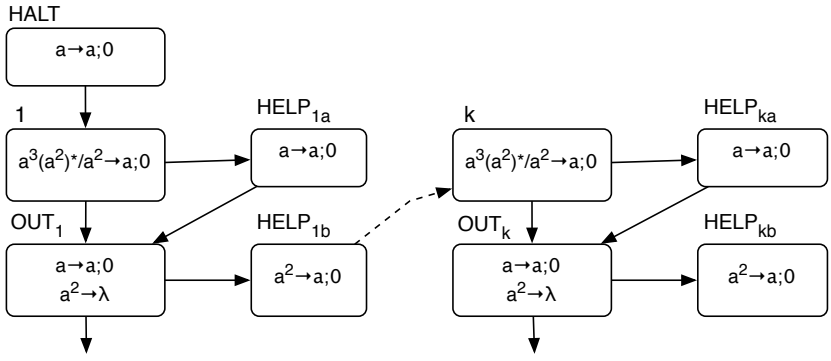


Fig. 6. Output module for reversal-bounded k-output unbounded SNP

nondeterministic choice module. The linking of $k-1$ nondeterministic choice modules allows us to nondeterministically choose to next execute one and only one instruction from the set $\{l_{j_1}, \cdots, l_{j_k}\}$. When the add module completes its computation, no spikes are left in the module allowing the instruction to be executed repeatedly.

The output module is initiated when instruction l_i : (HALT) is executed by sending a spike to neuron $HALT$. This neuron fires sending a single spike to neuron 1. Now the neuron contains an odd number of spikes causing the neuron to fire continuously (and decrementing the number of spikes by 2 at each step) until the neuron contains only a single spike. Each time the neuron spikes, a spike is sent to neuron $HELP_{1a}$ and neuron OUT_1. Neuron $HELP_{1a}$ spikes (sending a spike to neuron OUT_1) in the following step for each spike it receives. This guarantees that neuron OUT_1 will receive a single spike after neuron 1 spikes for the first time and after neuron 1 spikes for the last time. Neuron OUT_1 spikes at each of these two steps generating n_1. The two spikes of neuron OUT_1 are also sent to neuron $HELP_{1b}$ which fires after the second spike is received. This spike is sent to neuron 2 causing its spikes to be odd. We repeat the process until all the numbers $(n_1, \cdots, n_k)$ have been generated.

We now show that (2) implies (1). By Theorem 3, it is sufficient to show that any reversal-bounded k-output SNP can be simulated by a reversal-bounded k-output CM. Given a reversal-bounded k-output SNP Π, we construct a reversal-bounded k-output CM $\mathcal{M}$ to simulate Π. $\mathcal{M}$ has several counters, which are initially zero. k counters are the output counters, $G_1, \cdots, G_k$. For $1 \leq i \leq k$, when output neuron O_i in Π has generated the first spike a, output counter G_i starts incrementing at every step of the simulation. When O_i has generated the second spike a, G_i stops incrementing. When all the output counters have stopped incrementing, $\mathcal{M}$ enters the *Ending Phase*, which we will describe below.

Now we describe how the other counters of $\mathcal{M}$ are used to keep track of the number of spikes in the neurons during the computation. A bounded neuron is easy to simulate and does not need a counter. Since the regular expression

in a bounded neuron represents a finite set, the simulation of the neuron can be done in the finite control.

Now consider an unbounded neuron. For each unbounded rule $E/a^j \to a; 0$, the rule can be applied when the number of current spikes is covered by the expression E. Since E is a regular expression it can be written in the form $(a^{i_1}(a^{k_1})^*)+\cdots+(a^{i_t}(a^{k_t})^*)$ where $i, k \geq 0$, $t \geq 1$. For such a rule, we need t auxiliary counters and $2t$ buffers in the finite control (of sizes $i_1, \cdots, i_t, k_1, \cdots, k_t$). The first of these counters will store the number of spikes (n) minus i_1 currently in the neuron divided by k_1 (so the counter stores $(n - i_1)/k_1$). The second counter will store $(n - i_2)/k_2$ and so forth. The buffers and counters operate as follows.

Every time a spike enters the neuron in Π, the buffers in $\mathcal{M}$ corresponding to $a^{i_1}, \cdots, a^{i_t}$ are incremented. If any of these buffers are full, the buffer is left unchanged and the corresponding buffer in $a^{k_1}, \cdots, a^{k_t}$ is incremented. If one of these buffers is full, the buffer is emptied and the corresponding counter is incremented.

To simulate an unbounded neuron in Π firing, $\mathcal{M}$ nondeterministically picks an applicable rule. (A rule is applicable when some buffer a^{i_s} is full and buffer a^{k_s} is empty.) To fire the rule, we must remove j spikes from the neuron. To simulate this, we remove j from each of our buffers $a^{j_1}, \cdots, a^{j_t}$. If any of these buffers do not have enough to subtract j, we subtract what we can (until the buffer is zero) and then try to subtract one from the corresponding counter. If the counter contained at least 1, the buffer is refilled and the remainder of j is subtracted. If the counter is zero, the remainder of j is subtracted from the corresponding buffer a^i.

Finally, we note that Π is a "maximal parallel" system with respect to the number of neurons that can fire in any given step. $\mathcal{M}$ can easily keep track in its finite control (using the values of the bounded neurons and the values of the buffers and counters simulating the unbounded neurons) the maximal number of neurons fireable at any given step. It follows that $\mathcal{M}$ generates $Q(\Pi)$, and $\mathcal{M}$ is reversal-bounded.

Ending Phase: This phase is entered only when the k output counters have k values generated by the k output neurons of the SNP. $\mathcal{M}$ continues the simulation until Π halts. By definition, this happens when all the neurons, except for a specified subset R of neurons, have zero spike and those in R have two spikes. $\mathcal{M}$ checks that this configuration has been entered and enters an accepting state. □

Remark 1. We note that the theorem above holds even if we restrict the unbounded neurons in the SNP to have only the rule $a^3(a^2)^*/a^2 \to a; 0$.

4 Reversal-Bounded *k*-Train Unbounded SNPs

Now consider another restricted model called *reversal-bounded k-train SNP*. Such a system has only one output neuron. Again, there are two types of neurons: bounded and unbounded but reversal-bounded, as defined before. We say that

Π generates the k-tuple $(n_1, \cdots, n_k)$ if, starting from the initial configuration, the output neuron O generates the spike train $aa \cdots a$ with exactly $k+1$ outputted a's such that the interval between the i^{th} a and the $i+1^{st}$ a is n_i, and the system eventually halts.

Theorem 5. *The following statements are equivalent for any $Q \subseteq N^k$:*

1. *Q is semilinear.*
2. *Q can be generated by a reversal-bounded k-train unbounded SNP.*
3. *Q can be generated by a 1-reversal k-train unbounded SNP.*

Proof. Again (3) implies (2). The proof of (2) implies (1) is similar to the one in Theorem 4. We now show that (1) implies (3)

To prove that any semilinear set can be generated by a restricted k-train SNP only requires changing the output module from Theorem 4 since the add module can work in the same manner as before. The new output module can be seen in Figure 7. This module is initiated when instruction l_i : (HALT) is simulated

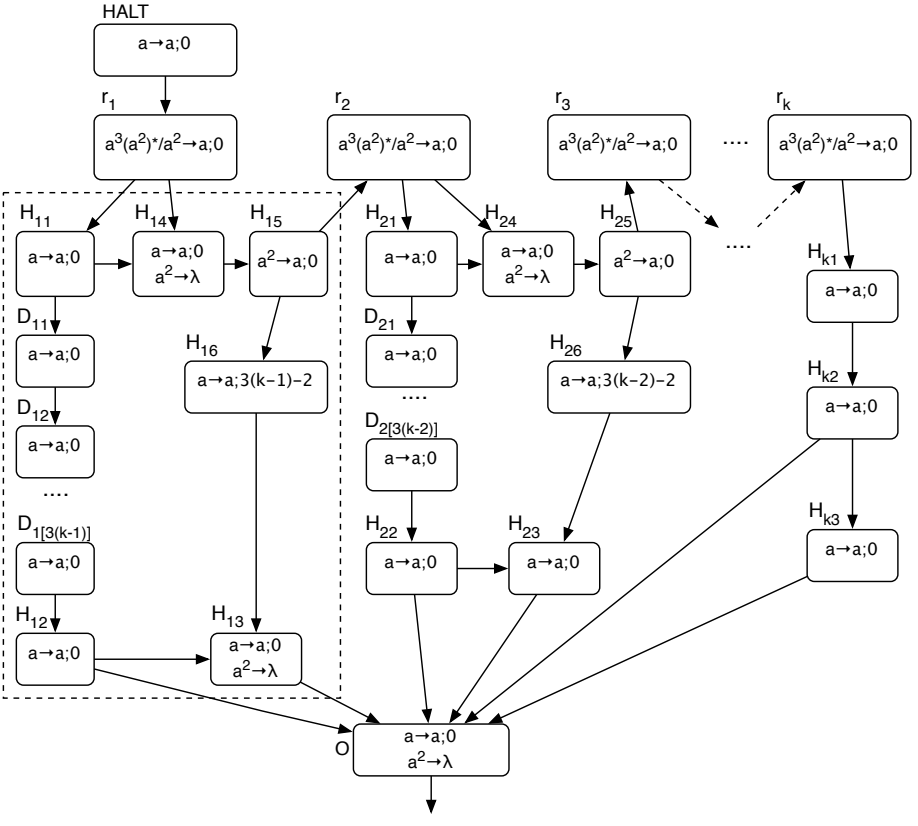


Fig. 7. Output module for reversal-bounded k-train unbounded SNP

sending a single spike to neuron $HALT$. This causes neuron $HALT$ to spike which sends a single spike to the neuron r_1 causing the neuron to contain an odd number of spikes. This causes the neuron to continuously spike (decreasing the number of spikes it contains by two at each step) until only a single a is contained in neuron r_1. Each time neuron r_1 spikes, a spike is sent to both neuron H_{11} and neuron H_{14}. This causes neuron H_{14} to spike after neuron r_1's initial spike and after its final spike. This allows a single spike to be sent to the second counter neuron once the first counter is done decrementing (with a delay of three steps). The delay of three steps between each counter is negated by the use of the delay neurons D_{11} to $D_{1[3(k-1)]}$.

The output neuron O receives spikes from the first counter through neurons H_{12} and H_{13}. Neuron H_{13} receives spikes from neuron H_{12}. This means that neuron O will first receive a single spike from neuron H_{12} which represents the first spike of neuron r_1 which causes O to spike. At each additional step, neuron O receives two spikes from both neuron H_{12} and neuron H_{13}. Unlike the proof of Theorem 4, neuron O never receives a single spike from neuron H_{13} because as soon as H_{12} no longer contains a spike, neuron H_{16} sends a second spike to H_{13} causing it to no longer be applicable (since it will contain two spikes). However, neuron O will receive a single spike designating the beginning time for n_2 which also happens to be the correct ending time for n_1.

The neurons designated by the dashed box in Figure 7 are repeated for each counter neuron with a few modifications. The number of delay neurons is $3(k-i)$ for each counter i. Also, the delay on the rule in each neuron H_{i6} needs to be changed so that the spike corresponds to the last spike in H_{13}. Therefore, this delay is $3(k-i)-2$ for each H_{i6}. This means that for counter k, no delay neurons are needed. Also, the last counter does not need to initiate any further counters nor should the single spike from neuron H_{k3} be thwarted. (This last spike is needed to send the final spike from neuron O.) Therefore, neurons H_{k4}, H_{k5}, and H_{k6} are not needed. □

Remark 2. Again, we note that the theorem above holds even if we restrict the unbounded neurons in the SNP to have only the rule $a^3(a^2)^*/a^2 \to a; 0$.

5 *k*-Train Bounded SNPs

We will give a characterization of k-train SNPs all of whose neurons are bounded. A strictly monotonic k-output CM is *sequential* if for $1 \leq r < k$, counter $r+1$ can (and must) only start incrementing when counter r has stopped incrementing aft er having been incremented. Hence, a sequential strictly monotonic k-output CM (simplified to *sequential k-output CM*) has simplified rules of the forms

$l_i : (\texttt{BEGIN}(1), l_{i_1}, \cdots, l_{i_t})$
$l_i : (\texttt{BEGIN}(r+1), \texttt{END}(r), l_{i_1}, \cdots, l_{i_t})$ for $1 \leq r < k$
$l_i : (\texttt{END}(k), l_{i_1}, \cdots, l_{i_t})$
$l_i : (\texttt{CONTINUE}, l_{i_1}, \cdots, l_{i_t})$
$l_i : (\texttt{HALT})$

For a set $Q \subseteq N^k$, define the language (over k symbols $a_1, \cdots, a_k$), $L_Q = \{a_1^{n_1} \cdots a_k^{n_k} \mid (n_1, \cdots, n_k) \in Q\}$. Clearly, Q is generated by a sequential k-output CM if and only if L_Q is a regular set, i.e., accepted by a nondeterministic finite automaton (NFA).

Theorem 6. *A set $Q \subseteq N^k$ is generated by a k-train bounded SNP if and only if it is generated by a sequential strictly monotonic k-output CM.*

Proof. Given a k-train bounded SNP generating Q, it is straightforward to construct a sequential k-output CM generating Q.

For the converse, let $\mathcal{M}$ be a sequential k-output CM $\mathcal{M}$ generating Q. We construct a k-train bounded SNP generating Q using the same techniques as in Theorem 2 with a few slight changes. Since each of the instructions in $\mathcal{M}$ is just a simplified version of the instructions in Theorem 2, clearly the simulation can be created in the same fashion. There are two main differences in creating Π.

First, for each instruction of the form $l_i : (\texttt{BEGIN}(r+1), \texttt{END}(r), l_{i_1}, \cdots, l_{i_t})$, we instead simulate the instruction $l_i : (\texttt{END}(r), l_{i_1}, \cdots, l_{i_t})$. The reason for this is due to the nature of a spike train which only requires $k+1$ output spikes

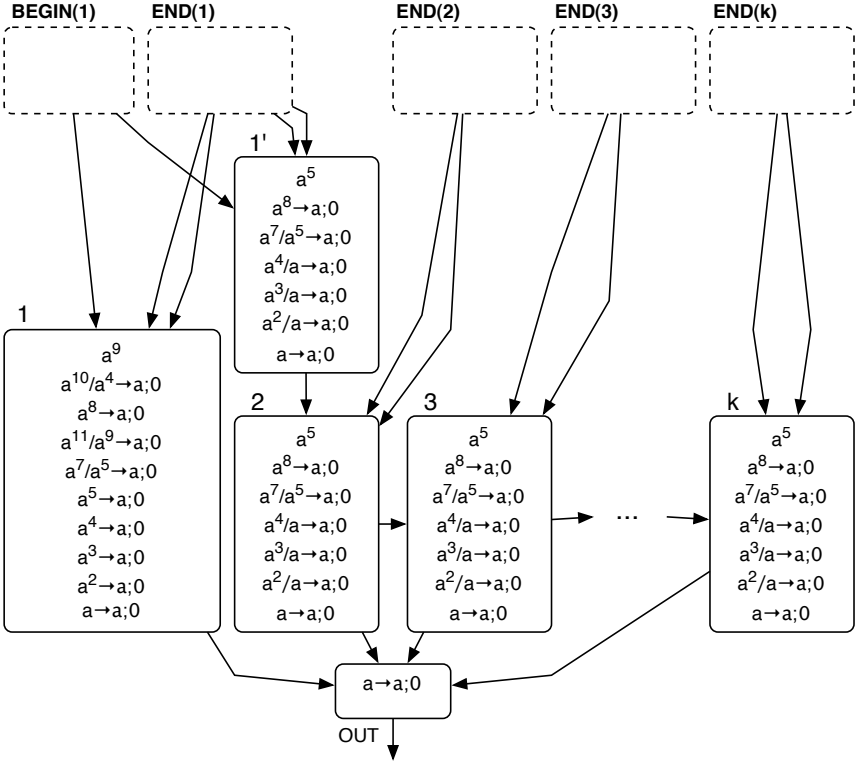


Fig. 8. Bounded train SNP output module

(rather than the previous $2k$ output spikes) since each middle spike represents both the end of output r and the beginning of output $r+1$.

The second difference is in how the output neuron is triggered. For this construction, the changes in the output are reflected in Figure 8. The new manner of output requires two new neurons ($1'$ and OUT) and different rules in neurons $1, \cdots, k$. Since the output is a train, we only have one output neuron labeled OUT which first spikes when instruction $l_i : (\texttt{BEGIN}(1), l_{i_1}, \cdots, l_{i_t})$ is executed. OUT spikes a second time when the instruction $l_i : (\texttt{BEGIN}(2), \texttt{END}(1), l_{i_1}, \cdots, l_{i_t})$ is executed and it continues to spike for each consecutive BEGIN/END instruction until it spikes for the $k+1$st time after executing the instruction $l_i : (\texttt{END}(k), l_{i_1}, \cdots, l_{i_t})$.

The simulating SNP system Π must check that the order of instructions is correct (meaning instruction $l_i : (\texttt{BEGIN}(r+1), \texttt{END}(r), l_{I_1}, \cdots, l_{i_t})$ is executed before instruction $l_{i'} : (\texttt{BEGIN}(r+2), \texttt{END}(r+1), l_{i'_1}, \cdots, l_{i'_t})$ and after instruction $l_{i''} : (\texttt{BEGIN}(r), \texttt{END}(r-1), l_{i''_1}, \cdots, l_{i''_t})$). Π must also check that each BEGIN/END instruction is only executed once. The rules given in Figure 8 guarantee that if the computation of $\mathcal{M}$ is invalid, then the simulated computation in Π will output more than $k+1$ spikes (which is also invalid). □

Corollary 1. *For $k \geq 2$, k-train bounded SNPs are strictly weaker than k-output bounded SNPs.*

6 Conclusion

The results in this paper can be summarized as follow. For $k \geq 2$:

sequential strictly monotonic k-output CMs
= k-train bounded SNPs
< k-output bounded SNPs
= strictly monotonic k-output CMs
< reversal-bounded k-output CMs
= reversal-bounded k-output unbounded SNPs
= reversal-bounded k-train unbounded SNPs
= semilinear sets over N^k.

In the above '=' means equivalent, and '<' means weaker.

Suppose we augment a reversal-bounded k-output unbounded SNP with one unbounded free neuron. Thus, one neuron is unbounded with no reversal bound and all the other neurons are reversal-bounded. Call such a system *reversal-bounded + 1-free k-output unbounded SNP*. This model of SNP can be simulated by a reversal-bounded k-output CMs augmented with one free (i.e., unrestricted counter). It is known that this model is equivalent to one with only reversal-bounded counters [4]. Hence such CMs generate only semilinear sets. Thus, we can add "= reversal-bounded k-output CMs + 1 free counter" at the end of the above results. For the case $k = 1$, all the models above are equivalent, and they generate exactly the semilinear sets over N^1 (i.e., regular sets over a^*).

It follows that many closure properties (e.g., union, intersection, and complementation) hold for sets generated by the SNP models above. Similarly, many standard decision problems (e.g., membership, containment, and equivalence problems) are decidable.

References

1. W. Gerstner, W Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge Univ. Press, 2002.
2. S. Greibach: Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7, 3 (1978), 311–324.
3. T. Harju, O. Ibarra, J. Karhumaki, and A. Salomaa: Some decision problems concerning semilinearity and commutation. *Journal of Computer and System Sciences*, 65 (2002), 278–294.
4. O. Ibarra: Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25 (1978), 116–133.
5. O. Ibarra, A. Păun, Gh. Păun, A. Rodriguez-Paton, P. Sosik, and S. Woodworth: Normal forms for spiking neural P systems, submitted, 2006.
6. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308 (also available at [13]).
7. W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8, 1 (2002), 32–36.
8. W. Maass, C. Bishop, eds.: *Pulsed Neural Networks*, MIT Press, Cambridge, 1999.
9. M. Minsky: *Computation – Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.
10. Gh. Păun: *Membrane Computing – An Introduction*. Springer-Verlag, Berlin, 2002.
11. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, 17, 4 (2006), 975–1002.
12. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. Submitted, 2006.
13. The P Systems Web Page: `http://psystems.disco.unimib.it`.

A Membrane Algorithm for the Min Storage Problem

Alberto Leporati and Dario Pagani*

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano – Bicocca
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
leporati@disco.unimib.it,
dario.pagani@gmail.com

Abstract. Min Storage is an NP–hard optimization problem that arises in a natural way when one considers computations in which the amount of energy provided with the input values is preserved during the computation. In this paper we propose a polynomial time membrane algorithm that computes approximate solutions to the instances of Min Storage, and we study its behavior on (almost) uniformly randomly chosen instances. Moreover, we compare the (estimated) coefficient of approximation of this algorithm with the ones obtained from several other polynomial time heuristics. The result of this comparison indicates the superiority of the membrane algorithm with respect to many other traditional approximation techniques.

1 Preliminaries

Membrane systems (also known as *P systems*) were introduced in [7] as a new class of distributed and parallel computing devices, inspired by the structure and functioning of living cells. The basic model consists of a hierarchical structure composed by several membranes, embedded into a main membrane called the *skin*. Membranes divide the Euclidean space into *regions*, that contain some *objects* (represented by symbols of an alphabet) and *evolution rules*. Using these rules, the objects may evolve and/or move from a region to a neighboring one. The rules are applied in a nondeterministic and maximally parallel way: all the objects that may evolve are forced to evolve. A *computation* starts from an initial configuration of the system and terminates when no evolution rule can be applied. The result of a computation is the multiset of objects contained into an *output membrane* or emitted from the skin of the system.

In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems. For details, and a systematic introduction on the subject, we refer the reader to [9]. The latest information about P systems can be found in [11].

* This research was supported by the European Research Training Network Segravis.

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 443–462, 2006.

In [6], Nishida has proposed a new type of approximation algorithms for optimization problems, named *membrane algorithms*. A membrane algorithm operates on a particular type of P system, which is a linear collection of separated regions determined by nested membranes. Each region contains a number of candidate solutions, and a local optimization algorithm. At each computation step, the local optimization algorithms which occur in the system are concurrently executed on the currently available solutions. New candidate solutions are produced in each region as a result; the best and the worst of them are sent to the immediately inner and immediately outer region, respectively. By repeating this process, a good solution will likely appear in the innermost region after a suitable number of computation steps. The algorithm terminates after a prefixed number of iterations has been performed, or some halting condition is verified (such as, for example, the solution(s) of the innermost region is (resp., are) not changed for a predetermined number of steps). In [6], a membrane algorithm which computes approximate solutions to the instances of the Travelling Salesman Problem (TSP) is described; moreover, the results of some computer experiments are presented, showing that this algorithm is indeed a good approximation heuristic for TSP.

In this paper we elaborate after Nishida's membrane algorithm for TSP, and we propose a new membrane algorithm for another NP–hard optimization problem, Min Storage [5]. This problem arises in a natural way if we consider *conservative computations*, that is, computations in which the amount of energy associated with the input values is first preserved during the computation of the output values, and then it is completely returned with them. In [5], it has been proved that Min Storage is strongly NP–hard, and that it is 2–approximable. This means that there exists a polynomial time algorithm that, for every instance $\mathcal{E}$ of Min Storage, returns a feasible solution $\mathrm{sol}(\mathcal{E})$ which is at most the double of the optimal solution $\mathrm{opt}(\mathcal{E})$. In this paper we present the results of some computer experiments in which the new membrane algorithm for Min Storage is compared with several "classical" polynomial time heuristics. As we will see, the membrane algorithm performs considerably well (with an estimated coefficient of approximation which is well under 2), especially when the number of membranes and the number of iterations in the algorithm are sufficiently large (where the term "sufficiently" has been experimentally determined).

The paper is organized as follows. In Section 2 we recall the definition of Min Storage and some of its properties. In Section 3 we present a membrane algorithm to solve the problem. Precisely, we propose two versions: MA4MS (Membrane Algorithm for Min Storage) which uses a particular kind of crossover and mutation to produce new candidate solutions, and MA4MS LS, which uses only a simple local search. In Section 4 we discuss a method that allows to generate random instances of Min Storage with an (almost) uniform distribution of probability. Section 5 illustrates the results of some computer experiments which have been performed on MA4MS and MA4MS LS. As we will see, these results led us to abandon MA4MS and to use only the version with the local search for further investigation. In Section 6 we describe several "classical" polynomial time

heuristics for Min Storage. Section 7 illustrates the results of other computer experiments, which have been performed to compare the performance (in terms of average coefficient of approximation) of our membrane algorithm against the above classical heuristics. Section 8 concludes the paper.

2 The Problem

Let us first introduce the problem we want to solve. As stated in the Introduction, this problem comes from the study of *conservative* (that is, *energy preserving*) computations. We refer the interested reader to [5] for two possible interpretations of the problem in this setting.

Let $\mathcal{E} = \langle e_1, e_2, \ldots, e_k \rangle$ be a finite sequence of integer numbers. For a fixed $i \in \{1, 2, \ldots, k\}$, the *i-th prefix sum of* $\mathcal{E}$ is the value $\sum_{j=1}^{i} e_j$. Let C be a positive integer; we say that $\mathcal{E}$ is *C–feasible* if for each $i \in \{1, 2, \ldots, k\}$ the i-th prefix sum of $\mathcal{E}$ is in the closed interval $[0, C]$.

Problem 1. Name: ConsComp.

- Instance: a set $\mathcal{E} = \{e_1, e_2, \ldots, e_k\}$ of integer numbers such that $e_1 + e_2 + \ldots + e_k = 0$, and an integer number $C > 0$.
- Question: is there a permutation $\pi \in S_k$ (the symmetric group of order k) such that the sequence $e_{\pi(1)}, e_{\pi(2)}, \ldots, e_{\pi(k)}$ is C–feasible? □

The fact that the resulting sequence $e_{\pi(1)}, e_{\pi(2)}, \ldots, e_{\pi(k)}$ is C–feasible can be explicitly written as:

$$0 \leq \sum_{j=1}^{i} e_{\pi(j)} \leq C \qquad \forall\, i \in \{1, 2, \ldots, k\}$$

The following theorem, which has been proved in [5], shows that it is very unlikely that a polynomial time algorithm exists that correctly classifies every instance of ConsComp as positive or negative.

Theorem 1. ConsComp *is* NP*–complete.*

The ConsComp problem naturally leads to the formulation of the following optimization problem.

Problem 2. Name: Min Storage.

- Instance: a set $\mathcal{E} = \{e_1, e_2, \ldots, e_k\}$ of integer numbers such that $e_1 + e_2 + \ldots + e_k = 0$.
- Solution: a permutation $\pi \in S_k$ such that $\sum_{j=1}^{i} e_{\pi(j)} \geq 0$ for each $i \in \{1, 2, \ldots, k\}$.
- Measure: $\max_{1 \leq i \leq k} \sum_{j=1}^{i} e_{\pi(j)}$. □

Informally, the output of Min Storage is the minimum value of C for which there exists a permutation $\pi \in S_k$ such that the sequence $e_{\pi(1)}, e_{\pi(2)}, \ldots, e_{\pi(k)}$ is C–feasible. Notice that a trivial upper bound for the value of C is:

$$\sum_{i \in \{1,2,\ldots,k\} \,:\, e_i > 0} e_i = \frac{1}{2} \sum_{i=1}^{k} |e_i|$$

while a trivial lower bound is $\max_{1 \leq i \leq k} |e_i|$.

It is immediately seen that Min Storage is in the class NPO [1, page 27]. In fact, checking whether some given integers $e_1, e_2, \ldots, e_k$ sum up to zero can be trivially done in polynomial time; each feasible solution has linear length and besides it can be verified in polynomial time whether a given permutation $\pi \in S_k$ is a feasible solution; finally, the measure function can be computed in polynomial time. Since the underlying decision problem ConsComp is NP–complete, we can immediately conclude that Min Storage is NP–hard [1, page 30]. Just like the ConsComp decision problem, this means that it is very unlikely that a polynomial time algorithm exists that gives the correct solution to every instance of Min Storage.

Since the Min Storage problem is NP–hard, a natural question is how well its optimal solutions can be approximated in polynomial time. Precisely, we ask ourselves whether there exists a PTAS (Polynomial Time Approximation Scheme) or even a FPTAS (Fully Polynomial Time Approximation Scheme) for Min Storage. Concerning these questions, in [5] it has been proved that ConsComp is NP–complete in the strong sense, by showing a polynomial reduction from 3–Partition [2, page 224], a well known strongly NP–complete problem. As a consequence, Min Storage is strongly NP–hard, and thus it doesn't admit a FPTAS [1, page 116]. The next natural question is whether there exists a PTAS for Min Storage; this possibility is currently under investigation.

In [5] it has also been proved that the algorithm shown in Figure 1 is a 2–approximation algorithm for Min Storage. The proof derives from the fact that, denoted by M the value $\max_{1 \leq i \leq k} |e_i|$, the variable st (that records the energy currently stored into the system) assumes values only from the interval $[0, 2M - 1]$. The variable max, which contains the value returned at the end of the computation, records the maximum of the values assumed by st into the subinterval $[M, 2M - 1]$. Since the optimal solution cannot be less than M, the value returned by the algorithm is at most the double of the optimal solution. A direct inspection of the pseudo–code reveals that the time complexity of the algorithm is linear with respect to k, the length of the input sequence. Hence, Min Storage is in the class APX of problems which admit a constant factor polynomial time approximation algorithm.

3 A Membrane Algorithm for Min Storage

Let us now introduce a membrane algorithm that produces approximate solutions to any instance of Min Storage. As stated in the Introduction, the

```
APPROX MIN STORAGE(ℰ)
M ← max_{1≤i≤k} |e_i|
E_p = E_n = ∅
for i ← 1 to k
    do if e_i ≥ 0
          then E_p = E_p ∪ {e_i}
          else E_n = E_n ∪ {e_i}
max ← st ← 0
while E_p ≠ ∅
    do if st < M
          then x ← an element of E_p
               st ← st + x
               if st > max then max ← st
               E_p = E_p \ {x}
          else x ← an element of E_n
               st ← st + x
               E_n = E_n \ {x}
return max
```

Fig. 1. Pseudocode of a 2–approximation algorithm for MIN STORAGE

algorithm is based on a structure composed by nested membranes. Each of the regions determined by the membranes contains a certain number of candidate solutions, and a local optimization algorithm. Formally, let m be the number of nested membranes, and let 0 and $m-1$ be the innermost and the outermost regions, respectively. Just like in the membrane algorithm for TSP, proposed by Nishida in [6], we put one candidate solution in region 0 and two candidate solutions in the other regions.

Another fundamental component of the system is the *transport* mechanism, that allows candidate solutions to move to the immediately inner or to the immediately outer region. The idea underlying the algorithm is to move good solutions towards the innermost region, and bad solutions towards the outermost region. When the computation halts, the best candidate solution produced by the system is thus contained into the innermost region, which is by definition the region in which the output is observed at the end of the computation.

A first difficulty in adapting the TSP membrane algorithm proposed by Nishida to the MIN STORAGE problem is that, differently from TSP, not all the permutations of the elements $\mathcal{E} = \{e_1, e_2, \ldots, e_k\}$ given in the instance give rise to feasible solutions. This is due to the fact that in a feasible solution π of MIN STORAGE all prefix sums are non negative; clearly, this property is not preserved if we exchange two randomly chosen elements of the solution. To overcome this difficulty, we have slightly modified the measure function associated with MIN STORAGE as follows:

$$F(\pi) = \begin{cases} \max\limits_{1 \le i \le k} \sum\limits_{j=1}^{i} e_{\pi(j)} & \text{if } \sum_{j=1}^{i} e_{\pi(j)} \ge 0 \text{ for all } i \in \{1, \ldots, k\} \\ \sum\limits_{i=1}^{k} |e_i| - \mathrm{NumVPS}_\pi & \text{otherwise} \end{cases}$$

where NumVPS_π is the number of non negative (that is, valid) prefix sums determined by π. In this way, all feasible solutions get a lower measure with respect to non feasible solutions. Moreover, every permutation can be measured, and we can also choose what among two non feasible solutions to prefer: the one which has the lowest number of negative prefix sums. As an alternative

approach, we could impose to work only with feasible solutions (discarding non feasible ones when they appear), and adopt the usual measure function for MIN STORAGE. However, since the probability to generate a non feasible solution is very high, this approach has been considered infeasible from a computational point of view.

The structure of the algorithm is analogous to the one proposed by Nishida for TSP. Given an instance $\mathcal{E}$ of MIN STORAGE, the algorithm works as follows:

1. put one random solution in region 0, and two random solutions in every region from 1 to $m-1$;
2. repeat the following steps d times:
   (a) in each region, apply the local optimization algorithm to produce new candidate solutions;
   (b) for every region $i \in \{1, 2, \ldots, m-1\}$, send the best among the solutions contained in the region (both old and new) to region $i-1$ (that is, towards the interior of the system). Similarly, for all $i \in \{0, 1, \ldots, m-2\}$ send the worst solution of region i to region $i+1$;
   (c) in each region $i \in \{1, 2, \ldots, m-1\}$, remove all solutions but the best two. In region 0, remove all solutions but the best one;
3. return the solution contained in region 0 as the output of the algorithm.

With respect to the membrane algorithm for TSP, we have used different local optimization algorithms. Precisely, for region 0 we have used a kind of local search: given a solution π, this operation explores the solutions which can be found in its neighborhood (which depends upon a specified element of π); if one of such solutions has a better measure than π, then it substitutes π. Formally, the neighborhood of π is defined as follows.

Definition 1. *Let $\pi \in S_k$ be a candidate solution, and let $\alpha \in \{1, 2, \ldots, k\}$. The* neighborhood $Neigh(\pi, \alpha)$ *of π, with respect to position α, is the set of $k-1$ solutions defined as follows:*

$$Neigh(\pi, \alpha) = \bigcup_{i \neq \alpha} \{\pi_{i,\alpha}\}$$

where $\pi_{i,\alpha}$ is the solution obtained from π by exchanging the elements in positions i and α.

The local search in region 0 is thus executed as follows:

LocalSearch4MS(π, α)

```
Best ← π
min ← F(π)
for i ← 1 to k − 1 do
    π′ ← select an element from Neigh(π, α)
    if F(π′) < min
        then min ← F(π′)
```

$Best \leftarrow \pi'$
$Neigh(\pi, \alpha) = Neigh(\pi, \alpha) \setminus \{\pi'\}$
`return` $Best$

In order to improve the probability to generate feasible solutions, the position α with respect to which we build the neighborhood of π is chosen as the first position for which the corresponding prefix sum is negative; if all prefix sums are non negative, then α is chosen at random in the set $\{1, 2, \ldots, k\}$.

In a first version of our membrane algorithm, that we have called MA4MS (Membrane Algorithm for Min Storage), we have tried to use a kind of *crossover* operation between candidate solutions as a local optimization algorithm for regions $1, 2, \ldots, m-1$. The idea, very well known in the domain of *genetic algorithms*, is to start from two solutions A and B, cut them in the same position and then recombine them as shown on the left of Figure 2. However, if we apply this operation to permutations, it is very likely that we obtain sequences in which some elements are missing and some are repeated; that is, in general we do not obtain two permutations as a result. Hence we have tried to use a variant of the standard crossover operation, named *partially matched crossover* (or PMX, for short) [4]. Just like the standard crossover, PMX operates on two permutations, say π and π'. This time, however, the two permutations are not recombined; rather, two "cutpoints" are randomly selected (let us call them l and r, respectively, with $l \leq r$) and then the elements of π are permuted according to positions $\pi'(l), \pi'(l+1), \ldots, \pi'(r)$. Similarly, the elements of π' are permuted according to positions $\pi(l), \pi(l+1), \ldots, \pi(r)$. As an example, let us assume that $\pi = \langle 2, 3, 1, 7, 4, 6, 5 \rangle$, $\pi' = \langle 5, 1, 7, 6, 4, 2, 3 \rangle$, $l = 2$ and $r = 5$. This situation is depicted on the right of Figure 2. Now, in both permutations π and π' the elements 3 and 1 are exchanged, then the elements 1 and 7 are exchanged, and so on. The pseudocode of the PMX operation is the following:

PMX(π, π')
$l, r \leftarrow$ random$(1, \ldots, k)$
`if` $l > r$ `then` exchange l and r
`for` $i \leftarrow l$ `to` r `do`
    find the position $j \in \{1, 2, \ldots, k\}$ such that $\pi(j) = \pi'(i)$
    find the position $j' \in \{1, 2, \ldots, k\}$ such that $\pi'(j') = \pi(i)$
    exchange $\pi(i)$ and $\pi(j)$
    exchange $\pi'(i)$ and $\pi'(j')$
`return` π, π'

Once two new solutions have been generated using the PMX operation, a "mutation" operation is applied on each of them with the probability $p = \frac{i}{m}$, which is directly proportional to the depth of the region into the system. This means, in particular, that the mutation is never performed in the innermost region, whereas it is almost always applied in the outermost region. This operation simply chooses two positions in a random way (according to a uniform probability distribution) and then exchanges the elements in such positions.

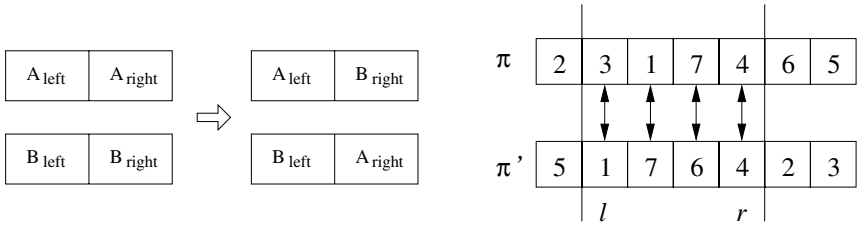


Fig. 2. The standard crossover operation (left) and the first step in partially matched crossover (right)

As we will see later, we have also considered a second version of the above membrane algorithm, in which no PMXs and no mutations are performed. Instead, LocalSearch4MS is used as the local optimization algorithm in every membrane of the system. We have called such variant MA4MS LS.

4 Generating Random Instances of Min Storage

We have performed some computer experiments on randomly chosen instances of Min Storage, in order to study the behavior of our membrane algorithm. All the random choices made during the experiments were performed according to the discrete uniform distribution. Hence the first problem we faced was to generate random instances for Min Storage in a uniform way. Formally, we can state the problem as follows.

Problem 3. Let $e_1, e_2, \ldots, e_k$ be independent variables uniformly distributed over the set of integers from the interval $[-M, M]$: how can we extract in a uniform way those k–tuples for which $e_1 + e_2 + \ldots + e_k = 0$? □

A possible solution to this problem could be to extract each element e_i and to discard the entire k–tuple if the sum is not zero; however the probability of success, $\text{Prob}\left[\sum_{i=1}^{k} e_i = 0\right]$, is fairly small. In order to compute such probability we observe that the distribution of the sum of k discrete independent uniformly distributed random variables is a k–th order convolution. Hence, the evaluation of the probability of success amounts to compute how many k–tuples with elements in $[-M, M]$ whose sum is zero we can build. To the best knowledge of the authors, this calculation seems to require the examination of an exponential number of k–tuples, and thus it is not feasible. As a consequence, we can compute an estimate of the probability of success by approximating the distribution of the random variable $Y = e_1 + e_2 + \ldots + e_k$ with an appropriate normal distribution.

First of all, let us compute the mean and variance of each random variable e_i. Since e_i is uniformly distributed over the interval $[-M, M]$ of integers, it holds:

$$\mathrm{E}[e_i] = \sum_{x=-M}^{M} x \cdot \frac{1}{2M+1} = \frac{1}{2M+1} \sum_{x=-M}^{M} x = 0$$

and

$$\begin{aligned}\mathrm{var}[e_i] &= \mathrm{E}\left[e_i^2\right] - (\mathrm{E}[e_i])^2 = \mathrm{E}\left[e_i^2\right] \\ &= \sum_{x=-M}^{M} x^2 \cdot \frac{1}{2M+1} = \frac{2}{2M+1} \sum_{x=1}^{M} x^2 = \frac{M(M+1)}{3}\end{aligned}$$

For linearity we obtain:

$$\mathrm{E}[Y] = \sum_{i=1}^{k} \mathrm{E}[e_i] = 0$$

Since $e_1, e_2, \ldots, e_k$ are independent variables, the variance of their sum is the sum of their variances, hence:

$$\mathrm{var}[Y] = \sum_{i=1}^{k} \mathrm{var}[e_i] = k \cdot \frac{M(M+1)}{3}$$

A direct consequence of the Central Limit theorem is that we can approximate the distribution of Y with the normal distribution $N\left(0, k\frac{M(M+1)}{3}\right)$ having the same mean and variance. In our experiments we have considered $k = 100$ and $M = 10^6$; this means that $\mathrm{var}[Y] \approx 10^{14}$, and the probability of success is:

$$\mathrm{Prob}[Y = 0] \approx 6.91 \cdot 10^{-8}$$

As stated above, this is a very small value. On the other hand, let us notice that there is a bijective correspondence between the set of all k–tuples whose sum is zero and the set of all $(k-1)$–tuples whose sum is in the interval $[-M, M]$. This observation suggests that we could extract $k-1$ elements in a uniform way and check whether their sum is in the interval $[-M, M]$. If this is the case then we put $e_k = -\sum_{i=1}^{k-1} e_i$, thus producing an instance; otherwise, we discard the entire $(k-1)$–tuple and we try with a different set of $k-1$ elements. Now the probability of successfully generate an instance is $\mathrm{Prob}\left[-M \leq \sum_{i=1}^{k-1} e_i \leq M\right]$. Once again, we can approximate the distribution of the random variable $Z = \sum_{i=1}^{k-1} e_i$ with the normal distribution $N\left(0, (k-1)\frac{M(M+1)}{3}\right)$. Thus, if

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2x^2}\right) \qquad \text{and} \qquad \Phi(x) = \int_{-\infty}^{x} \phi(u)\,\mathrm{d}u$$

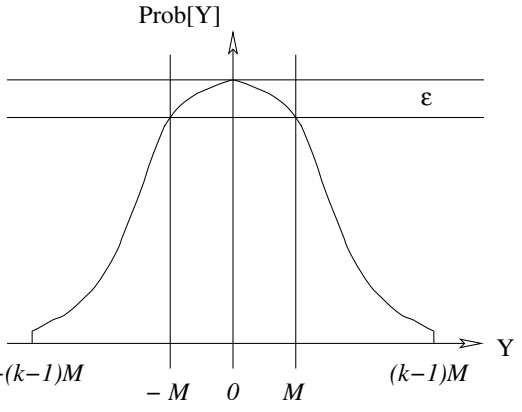


Fig. 3. Gaussian approximation of the distribution of $Z = \sum_{i=1}^{k-1} e_i$

are the probability density function and the cumulative distribution function of the standardized normal distribution $N(0, 1)$, it holds:

$$\text{Prob}\left[-M \leq \sum_{i=1}^{k-1} e_i \leq M\right] \approx \Phi\left(\frac{M}{\sigma}\right) - \Phi\left(-\frac{M}{\sigma}\right) = 2\Phi\left(\frac{M}{\sigma}\right) - 1$$

where $\sigma = \sqrt{(k-1)\frac{M(M+1)}{3}}$. For $k = 100$ and $M = 10^6$ we obtain:

$$\text{Prob}\left[-10^6 \leq \sum_{i=1}^{99} e_i \leq 10^6\right] \approx 0.138 \tag{1}$$

Intuitively, for fixed values of M and k we approximate the real distribution of Z (that can assume every integer value in the interval $[-(k-1)M, (k-1)M]$) with a normal distribution (see Figure 3), and we consider the portion of the curve contained into the vertical strip included between $-M$ and M. For growing values of k, such strip becomes small with respect to the entire curve, and thus the portion of the curve into the strip tends to become an horizontal segment. This means that we find k–tuples whose sum is zero with almost a uniform distribution. We can estimate the error due to the fact that the portion of curve into the strip is not horizontal by looking at the difference between the higher and the lower values it assumes in this interval:

$$\varepsilon = \text{Prob}\left[\sum_{i=1}^{k-1} e_i = 0\right] - \text{Prob}\left[\sum_{i=1}^{k-1} e_i = M\right]$$

For $k = 100$ and $M = 10^6$, the error is $\varepsilon \approx 1.04 \cdot 10^{-9}$. As a consequence, we can safely assume that our strategy produces k–tuples whose sum is equal to zero with a uniform probability distribution; moreover, as stated in (1), about 13.8%

of the times it will produce one of such k–tuples. A computer experiment has confirmed this last result.

Before looking at the experiments, let us recall the notion of coefficient of approximation. Let $c_A(\mathcal{E})$ be the value which is returned as a solution by a heuristic A for the instance $\mathcal{E}$ of the MIN STORAGE problem, and let $opt(\mathcal{E})$ be the optimal solution, that is, the value returned by the brute force algorithm that examines all possible feasible solutions. Then, the *coefficient of approximation* of algorithm A over the instance $\mathcal{E}$ is the value $app_A(\mathcal{E})$, where

$$app_A(\mathcal{E}) = \frac{|c_A(\mathcal{E})|}{opt(\mathcal{E})} \tag{2}$$

Note that $app_A(\mathcal{E})$ is always greater than or equal to 1, and that the closer it is to 1, the better the approximate solution is. We say that algorithm A has the *guaranteed coefficient of approximation* c if $app_A(\mathcal{E}) \leq c$, for every instance $\mathcal{E}$. For example, APPROX MIN STORAGE has a guaranteed coefficient of approximation equal to 2.

5 First Experiments with MA4MS

We have implemented the membrane algorithm MA4MS in the JAVA programming language. To simulate the parallel application of local optimization algorithms we have implemented them as *threads*, with a monitor that allows to synchronize the exchange of information between the regions of the system.

Then, we have performed some computer experiments to study the behavior of MA4MS on randomly chosen instances. To measure the performance of the algorithm we have computed an estimate of its coefficient of approximation (see also equation (2)), averaged on the number N of instances considered in the experiment:

$$app_{\text{MA4MS}} = \frac{1}{N} \sum_{i=1}^{N} \frac{F_i(\pi)}{opt_i}$$

where opt_i has been put equal to the optimal solution of the i-th instance in those experiments for which the length of the instances allowed to compute it. In the experiments for which the length of the instances did not allow to compute the optimal solution with the brute force approach, we have substituted it with the theoretical lower bound $\max_{1 \leq i \leq k} |e_i|$.

In the first experiment we have tested the behavior of MA4MS by running 10000 tests, each with randomly generated instances of increasing length (10, 20, 50 and 100). The number m of regions and the number d of iterations have been put equal to 10 and 50, respectively. The results are illustrated in Figure 4 (on the left). As we can see, the average coefficient of approximation grows together with k, the length of the instances, going well above the value 2 given by APPROX MIN STORAGE. This is probably due to the fact that the partially matched crossover is not able to differentiate the solutions initially assigned to the system. Indeed, with PMX, solutions that differ in a small number of

k	app_{MA4MS}	VARIANCE
10	1.2052383	0.0433753
20	1.7645406	0.1412564
50	3.0863457	0.6402221
100	4.6576763	1.8045398

k	app_{MA4MS}	VARIANCE
10	1.0875901	0.0139737
20	1.5258556	0.0582978
50	2.6124590	0.2444580
100	3.9430665	0.5004649

Fig. 4. Results obtained for MA4MS on 10000 tests, with $m = 10$ and $d = 50$ (on the left) and with $m = 30$ and $d = 150$ (on the right), for different lengths k of the instances

positions produce new solutions which are similar. As we can see on the right of Figure 4, this problem remains even if we raise the parameters m and d to 30 and 150, respectively.

For these reasons, we have abandoned our first version of the membrane algorithm and we have repeated the above experiments with the second version, MA4MS LS, in which LOCALSEARCH4MS is used as a local optimization algorithm in all regions, instead of PMX and mutations. In Figure 5 (left) we can see the results of the second experiment described above, with $m = 30$ and $d = 150$.

k	app_{MA4MS}	VARIANCE
10	1.0032719	0.0002333
20	1.0093292	0.0003654
50	1.0600094	0.0045419
100	1.1978124	0.0162296

m	d	app_{MA4MS}
10	20	2.1003992
30	60	1.4055032
50	100	1.2228035
80	150	1.1003962
150	200	1.1066577
300	500	1.0214561

Fig. 5. Results obtained with MA4MS LS, on: (left) 10000 tests, with $m = 30$ and $d = 150$, for different lengths k of the instances; (right) groups of 10 instances of length 100, and growing values of m and d

It is apparent that MA4MS LS obtains better results, and thus we will use it in the following to perform some comparisons with some "classical" heuristics specially crafted for MIN STORAGE. Let us note that, in this new version of the algorithm, the only "forces" that drive to a good solution are local search and the transport mechanism, that moves good solutions towards region 0 and bad solutions towards region $m - 1$. No other forces (crossover, mutations, etc.) are involved, and hence it is our opinion that this is a "true" membrane algorithm, in the original spirit of Membrane Computing.

Some computer experiments have also been performed to see how the average coefficient of approximation is affected by the number m of regions and the number d of iterations. Figure 5 (on the right) shows some results obtained on groups of 10 instances, each containing 100 elements, for growing values of m and d. Notice that $d \geq m$, so that a good solution has always the possibility to reach the innermost region. Figure 6 contains two plots of the average coefficient

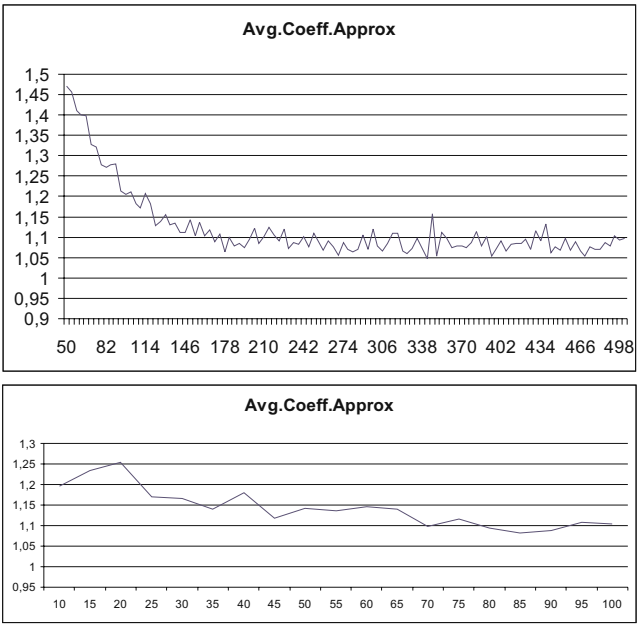


Fig. 6. Average coefficients of approximation obtained by letting vary d (up) and m (down) independently

of approximation with respect to growing values of d (up) and of m (down). All coefficients have been computed by performing 10 tests with instances of length 100. In the first experiment the value of m has been fixed to 80, and the value of d has been let vary in the interval $[50, 500]$; in the second experiment, instead, the value of d has been fixed to 150 and the value of m has been let vary in the interval $[10, 100]$.

We have also tried to increase the number of candidate solutions occurring in each region (but the innermost, which continues to have only one solution). As we can see in Table 1, the average coefficient of approximation decreases, but only slightly. Perhaps more interesting was to evaluate the *speed* of convergence of the

Table 1. Results obtained with MA4MS LS, by letting vary the number of candidate solutions in the regions (but the innermost). Each test contained 100 instances of length 100; $m = 30$ and $d = 160$.

Number of solutions	app_{MA4MS}	Variance
2	1.0609374	0.0022672
3	1.0392002	0.0018685
5	1.0231814	0.0007303
10	1.0145107	0.0005448

Table 2. Speed of convergence of MA4MS LS, with respect to the number of candidate solutions in the regions

Num. sol.	C.A. < 2	C.A. < 1.5	C.A. < 1.1	Avg. Gain
2	23	50	134	0.5065
3	17	38	111	0.3915
5	20	41	99	0.3355
10	15	37	93	0.2865

algorithm, with respect to the number of candidate solutions. Table 2 reports the results obtained with tests of 10 instances of length 100. The second, third and fourth column contain the number of steps needed (on average) to lower the average coefficient of approximation below 2, 1.5 and 1.1, respectively. Finally, in the last column we report the average of the gains obtained during each iteration of the algorithm. As we can see, when the number of candidate solutions grows the number of steps needed to obtain a good coefficient of approximation decreases. As a drawback, also the average gain decreases.

6 Some Heuristics for Min Storage

In this section we propose some "classical" polynomial time heuristics for solving Min Storage. Subsequently, we will run the same tests for both these heuristics and MA4MS LS, in order to compare the behavior of our membrane algorithm with more traditional approximation algorithms.

All the proposed algorithms have been implemented in the C programming language, to obtain the fastest execution times as possible. All lists have been implemented as arrays. We have associated a boolean flag to each element of the lists, indicating whether the element has to be considered as deleted or not, so that we can assume that the removal of a generic element $L[i]$ from a list L takes a constant time. As for sorting operations, we have assumed to use some comparisons–based optimal algorithm such as QuickSort or MergeSort, which take $\Theta(k \log k)$ time steps to sort k elements; in our experiments, we have indeed used the QuickSort routine included in the standard C libraries.

The first heuristic we consider is the *greedy* algorithm. This algorithm maintains a list L of elements to be considered. At the beginning of the execution L contains all the elements $\{e_1, e_2, \ldots, e_k\}$ of the instance. An integer variable st, initially set to 0, indicates the amount of energy currently stored into the gate. The algorithm repeats the following operations until L becomes empty: first it finds the minimum *positive* value of $st + \ell$, with $\ell \in L$, then it updates the value of st with $st + \ell$, and finally it removes ℓ from L. An integer variable $stmax$ records the maximum value reached by st; the value of $stmax$ at the end of the execution is the result returned by the greedy algorithm. It is easily seen that this algorithm can also be implemented as shown on the left side of

```
GREEDY(E)
L ← Sort(E)
st ← 0
stmax ← 0
while Length(L) > 0 do
    i ← 1
    while st + L[i] < 0 do
        i ← i + 1
    endwhile
    st ← st + L[i]
    stmax ← max{st, stmax}
    remove L[i] from L
endwhile
return stmax
```

```
MIN(E)
L_n ← negative values of E
L_p ← E \ L_n
sort L_p and L_n in increasing order
st ← 0
stmax ← 0
while Length(L_p) > 0 do
    st ← st + min(L_p)
    stmax ← max{st, stmax}
    remove min(L_p) from L_p
    while Length(L_n) > 0 and
                st + max(L_n) ≥ 0 do
        st ← st + max(L_n)
        remove max(L_n) from L_n
    endwhile
endwhile
return stmax
```

Fig. 7. Pseudocode for the GREEDY (on the left) and MIN (on the right) algorithms

Figure 7. From the inspection of the pseudocode it is clear that, under the hypotheses made above, the execution time of the whole algorithm is $\Theta(k^2)$.

Another heuristic is the MIN algorithm, whose pseudocode is shown on the right side of Figure 7. As we can see, at each iteration of the outer `while` loop the minimum of the remaining positive elements is chosen. For each positive element considered the inner `while` loop takes as many negative elements as possible, choosing the maximum of them (that is, the one with minimum absolute value) at each iteration. After an initial sorting, each element is considered only once during the execution of the two `while` loops; hence, the total execution time of the algorithm is $\Theta(k \log k)$. We have also considered a *dual* algorithm, which we have called MAX, where at each iteration of the outer loop the *maximum* of the remaining positive elements is chosen, whereas at each iteration of the inner loop the minimum of the remaining negative values is chosen.

Another variation is the MAXMINMAX algorithm, where at each iteration of the outer `while` loop the maximum of the remaining positive values is chosen, as in MAX. This time, however, there are two inner `while` loops: first we remove (as much as possible) the minimum negative elements, that is those with highest absolute value, and then we remove as much as possible the maximum elements. Also in this case there exists a dual algorithm, called MINMAXMIN, where at each iteration of the outer loop we remove the minimum of the remaining positive elements, and in the two inner loops we remove first the maximum and then the minimum of the remaining negative elements.

A further variation is given by algorithms MINMAXMINMAX and MAXMINMAXMIN. In the outer loop of these algorithms the maximum or the minimum of the remaining positive elements is alternately chosen; precisely, in the former

algorithm the first element chosen from the instance is the minimum of positive elements, whereas in the latter algorithm the maximum element of the instance is chosen first. The two inner loops are just like those of MaxMinMax and MinMaxMin; in particular, if the minimum of positive values has been chosen in the outer loop then we first remove the maximum negative elements and then the minimum ones, whereas we do the opposite if the maximum of positive elements was chosen. It is immediately seen that all the variations just exposed are uninfluent to the asymptotic execution time, that remains equal to $\Theta(k \log k)$.

Another approach to solve Min Storage is the Best Fit algorithm, shown in Figure 8. Best Fit assumes as a first estimate for the capacity of the gate

```
Best Fit(E)

L_n ← negative values of E
L_p ← E \ L_n
sort L_p and L_n in increasing order
est ← max_{1≤i≤k} |e_i|
st ← 0
while Length(L_p) > 0 do
    if st + min(L_p) > est then
        est ← st + min(L_p)
        st ← st + min(L_p)
        remove min(L_p) from L_p
    else
        for i ← Length(L_p) downto 1 do
            if st + L_p[i] ≤ est then
                st ← st + L_p[i]
                remove L_p[i] from L_p
            endif
        endfor
    endif
    for i ← 1 to Length(L_n) do
        if st + L_n[i] ≥ 0 then
            st ← st + L_n[i]
            remove L_n[i] from L_n
        endif
    endfor
endwhile
return est
```

Fig. 8. Pseudocode for the Best Fit algorithms

(denoted by *est* in the pseudocode) the theoretical lower bound $\max_{1 \le i \le k} |e_i|$. During the execution of the algorithm the estimate for the capacity is adjusted, by increasing it of the smallest possible amount. Precisely, at each iteration of the outer `while` loop we add to the internal storage some positive values from the instance, and then we add some negative values. Positive values of the instance are scanned from the maximum down to the minimum; each of them is added to the internal storage (and removed from the instance), unless the resulting value exceeds *est*. Analogously, negative values are scanned from the minimum to the maximum; each of them is added to the internal storage (and removed from the instance), unless the resulting value becomes negative. If at some point no positive value can be added — that is, if $st + \min(L_p) > est$, where st is the energy currently stored into the gate and $\min(L_p)$ is the minimum of the remaining positive elements — then we adjust the value of *est* by putting $est = st + \min(L_p)$. Now we can add $\min(L_p)$ to the internal storage and then try to add some negative elements. The result returned by the algorithm is the value of *est* at the end of the execution, that is, after all the elements of the

instance have been considered. A direct inspection of the pseudocode allows us to see that the execution time of BEST FIT is $\Theta(k^2)$.

7 Comparison Experiments

In this section we describe three computer experiments we have performed to compare the behavior of the proposed heuristics with MA4MS LS. Each instance was generated according to the random process described in Section 4. All the classical heuristics, as well as MA4MS LS (with $m = 300$ membranes and $d = 500$ iterations), have been executed on a number of instances, and for each algorithm we have computed its average coefficient of approximation as well as the corresponding variance. The results obtained during these experiments are illustrated in Figure 9.

In the first experiment we have generated 100 instances, each one containing 12 elements. The elements were chosen from the interval $[-10^6, 10^6]$ of integers. The small number and length of instances have been chosen in order to allow

ALGORITHM	app_A	VAR	app_A	VAR	app_A	VAR
Greedy	1.2052082	0.0615908	1.3844832	0.0664088	1.3948121	0.0468795
Min	1.3901304	0.0993216	1.7585659	0.0720077	1.6441352	0.0416634
Max	1.3804116	0.0979608	1.7533215	0.0733613	1.6424797	0.0449882
MaxMinMax	1.0863972	0.0143145	1.1051660	0.0055823	1.4993568	0.0784421
MinMaxMin	1.3901304	0.0993216	1.7585659	0.0720077	1.6441352	0.0416634
MaxMinMaxMin	1.1312751	0.0253605	1.1356385	0.0057628	1.5032439	0.0757982
MinMaxMinMax	1.1568342	0.0205104	1.1368552	0.0058827	1.1470192	0.0042202
Best Fit	1.0202729	0.0043468	1.0072024	0.0017742	1.1619727	0.0219509
MA4MS LS	1	0	1.0202449	0.0006565	1.0185239	0.0005584
	First experiment		Second experiment		Third experiment	

Fig. 9. Results obtained during the three computer experiments

the computation of optimal solutions through the "brute force" algorithm that examines all permutations in S_k. This means that the obtained results are the real average coefficients of approximation of the involved heuristics. Due to the length of instances, during the other two experiments we were not able to compute optimal solutions; hence, in those cases, in order to compute the coefficients of approximation we have used the theoretical lower bound $\max_{1 \leq i \leq k} |e_i|$ as the optimal solution, thus obtaining upper bounds to the real coefficients. Indeed, the first experiment was conceived to compare these upper bounds with the real coefficients of approximation, although computed over very small instances. As we can see from the tables, the values obtained are pretty similar.

In the first experiment, among the traditional heuristics BEST FIT has obtained the best average coefficient of approximation, and also the smallest variance; this means that it frequently finds a good solution. On the other hand, the membrane algorithm has always found the optimal solution. This is probably due

to the fact that, since the number of regions is high with respect to the length of the instances, then the probability that an optimal solution is produced during the initial generation of candidate solutions is very high. Further, a relatively high number of iterations in the algorithm allows such optimal solution to reach the innermost membrane before the algorithm halts.

In the second experiment we have generated 100000 instances of 100 elements, each taken from the interval $[-10^6, 10^6]$ of integers. As we can see in Figure 9, for traditional heuristics we have obtained results similar to those of the first experiment. MA4MS LS has obtained both a low average coefficient of approximation and a (very) low variance; moreover, it performs better than almost all the traditional heuristics. However, the winner is Best Fit. An interesting observation is that Best Fit did not find a solution equal to $\max_{1 \leq i \leq k} |e_i|$ for only 4564 of the 100000 instances; since the optimal solution cannot be less than this value, this means that for at least 95.4% of instances Best Fit found the optimal solution. We can explain this result by saying that Best Fit performs so well because it has been intentionally conceived for Min Storage. We are currently investigating whether higher values for the parameters m and d would lead to a better performance of MA4MS LS. Let us note, however, that even if this hypothesis should be true, the execution time of the algorithm would make us prefer again Best Fit, since it is very quick. Does this mean that we should forget MA4MS LS? The answer is negative, as shown by the next experiment.

For the third experiment, we have considered a variant of the Min Storage problem, where we have relaxed the requirement that the amount of energy stored into the gate at the beginning of the computation is zero. This corresponds to a natural extension of the notion of conservative computation, obtained by letting the gate to have a positive amount ε of energy stored at the beginning of the computation, and requiring that exactly the same amount ε of energy is stored into the gate at the end of the computation. When this situation occurs, we say that the computation is *ε–conservative*. Hence up to now we have dealt with 0–conservativeness. Clearly also the variant of Min Storage concerning ε–conservative computations (with $\varepsilon \geq 0$) is NP–hard, by the restriction property [2, page 63], since it contains Min Storage as a particular case.

In the third experiment we generated 100 instances, each one composed by 100 elements taken from the interval $[-10^6, 10^6]$ of integers. For each instance we ran the proposed algorithms, varying the initial energy ε from 0 to $\max_{1 \leq i \leq k} |e_i|$, with steps of 100. At first sight it may be surprising to see that Best Fit gives no more the best results: indeed, among the traditional heuristics MinMaxMinMax has both the lowest average coefficient of approximation and the lowest variance. It is our opinion that Best Fit does not perform better than MinMaxMinMax because the former algorithm starts by considering the elements of the instance from the greatest positive to the smallest positive element, each time taking the element if there is enough free storage into the gate; negative elements are considered only later. Of course, this may not be the optimal strategy, especially when the initial energy stored into

the gate is high with respect to gate capacity. The latter algorithm alternately chooses the minimum and the maximum of the positive elements remaining into the instance, and then it immediately considers negative elements: as a consequence, it has more chances to make the right choices. Some modifications to the Best Fit algorithm in order to perform better when there is a positive initial amount of energy into the gate are currently under consideration.

However, the absolute winner in this experiment is MA4MS LS. Once again it has a very low variance, and almost the same average coefficient of approximation as in the previous experiment; we can interpret this fact by saying that MA4MS LS is a *stable* algorithm, in the sense that its performance is not affected by small changes in the definition of the instances.

8 Conclusions

In this paper we have proposed some polynomial time approximation heuristics for Min Storage, a strongly NP–hard optimization problem that naturally arises in the context of conservative (that is, energy preserving) computations. One of the proposed heuristics is a membrane algorithm which was inspired by a previous work by Nishida [6].

We studied the behavior of all these heuristics on (almost) uniformly randomly chosen instances through several computer experiments. A first set of experiments allowed us to understand that a very simple local optimization algorithm, LocalSearch4MS, suffices to make the membrane algorithm obtain good solutions for Min Storage. We have called MA4MS LS the resulting algorithm. The results obtained from a second set of experiments suggest that Min Storage seems to be easy to solve on uniformly randomly chosen instances. In particular, one of the proposed heuristics, namely Best Fit, seems to perform very well when the initial energy stored into the gate is zero. Interestingly, the same heuristic is no more the best when the initial energy is positive.

If we look at the average coefficient of approximation obtained for MA4MS LS during the second set of experiments, we can see that we always obtain approximately the same value. Moreover, the low value obtained for the variance shows that the algorithm is also pretty stable, that is, its (average) behavior is not affected too much by small changes in the instances of the problem. If we compare this situation with the behavior of Best Fit, we can draw the following conclusions. Best Fit performs well since it is an algorithm which has been intentionally crafted for Min Storage; stated otherwise, it strongly reflects the structure of the problem. On the contrary, MA4MS LS is a *general* algorithm, that behaves in the same way independent of the problem on which it is applied.

Acknowledgements

We gratefully thank the anonymous referees, whose comments have helped us to improve a previous version of this paper.

References

1. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti–Spaccamela, M. Protasi. *Complexity and Approximation. Combinatorial Optimization Problems and Their Approximability Properties*. Springer–Verlag, 1999.
2. M.R. Garey, D.S. Johnson. *Computers and Intractability. A Guide to the Theory on NP–Completeness*. W.H. Freeman and Company, 1979.
3. G.V. Gens, E.V. Levner. Computational complexity of approximation algorithms for combinatorial problems. *Proceedings of the 8th International Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science 74, Springer–Verlag, Berlin, 1979, pp. 292–300.
4. D.E. Goldberg, R. Lingle. Alleles, Loci and the Traveling Salesman Problem. In *Proceedings of the International Conference on Genetic Algorithms*, 1985, pp. 154–159.
5. A. Leporati, C. Zandron, G. Mauri. Conservative Computations in Energy–based P systems. In G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa (Eds.) *Membrane Computing: 5th International Workshop, WMC 2004*, Milan, Italy, June 14–16, 2004, LNCS 3365, Springer–Verlag, 2005, pp. 344–358.
6. T.Y. Nishida. Membrane Algorithms. In R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.) *Membrane Computing: 6th International Workshop, WMC 2005*, Vienna, Austria, July 18–21, 2005, LNCS 3850, Springer–Verlag, 2006, pp. 55–66.
7. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 1(61):108–143, 2000. See also Turku Centre for Computer Science — TUCS Report No. 208, 1998.
8. Gh. Păun. Computing with Membranes. An Introduction. *Bulletin of the EATCS*, 67:139–152, February 1999.
9. Gh. Păun. *Membrane Computing. An Introduction*. Springer–Verlag, Berlin, 2002.
10. Gh. Păun, G. Rozenberg. A Guide to Membrane Computing. *Theoretical Computer Science*, 287(1):73–100, 2002.
11. The P systems Web page: `http://psystems.disco.unimib.it/`

P Systems with Symport/Antiport and Time

Hitesh Nagda[1], Andrei Păun[1,2], and Alfonso Rodríguez-Patón[2]

[1] Department of Computer Science/IfM, Louisiana Tech University
P.O. Box 10348, Ruston, LA 71272, USA
{hhn002, apaun}@latech.edu
[2] Universidad Politécnica de Madrid - UPM, Facultad de Informática
Campus de Montegancedo S/N, Boadilla del Monte, 28660 Madrid, Spain
arpaton@fi.upm.es

Abstract. We consider symport/antiport P systems using the time as the support for the output of a computation. We describe and study the properties of "timed symport/antiport systems", showing that this new model of membrane systems based on time has more power/flexibility, and thus allows us to improve previous universality results. We were able to improve or match the best results concerning the symport/antiport systems which consider the output as originally defined as the number of molecules found in a pre-defined elementary membrane in the halting configuration of the system.

1 Introduction

We continue the work on symport/antiport P systems which were considered in a series of recent papers. We refer the interested reader to [1], [2], [5], [6], [11], for basic definitions and results in this area. Briefly, the systems that we consider in this paper extend the original definition by using the paradigm of time as the output of a computation as previously introduced in [4] and [8]. The idea originates in [12] as Problem W; the novelty is that instead of the "standard" way to output, like the multiplicities of objects found at the end at the computation in a distinguished membrane as it was defined in the model from [11], it seems more "natural" to consider certain *events* (i.e., configurations) that may occur during a computation and to relate the output of such a computation with the time interval between such distinguished configurations. Our system will compute a set of numbers similarly with the case of "normal" symport/antiport systems as defined in [11], but the benefit of the current setting is that the computation and the observance of the output are now close to the biology and to the tools used for cell biology (fluorescence microscopy, FACS). The model of the "timed" P system that we investigate here is the symport/antiport P system. We note that such a "timed" approach could be applied also to other types of P systems. Actually the spiking neural P systems ([9], [14]) use a similar idea: the output of such a system is the time elapsed between two spikes of a pre-defined "output" neuron. Going back to the symport/antiport model, we are studying another way of viewing the output of such a system; the motivation comes from the fact

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 463–476, 2006.

that cells can become fluorescent if, for example, some types of proteins with fluorescence properties are present in the cells. Such a fluorescent "configuration" of a cell will be the configuration that starts the clock used for the output. Even more interesting (making our definition a very natural way of viewing the output of a system) is the fact that there are tools currently used by researchers in cell biology that can detect the fluorescence of each cell individually. Such an automated technique for viewing the output of a computation using cells is highly desirable since it holds the promise of fast readouts of the computations with low error rates in the readout.

2 Timed Symport/Antiport Systems

We will use a modified definition from the one given in [11]: instead of specifying the output region where the result of the computation will be "stored" in a halting computation, we specify two configurations (we can call them also relations) C_{start} and C_{stop} (which are described by regular languages) that need to be satisfied by the multisets of objects in the membrane structure at two different times during the computation. We restrict the description of C_{start} and C_{stop} to regular languages, (each word from the language representing a possible configuration of the system that satisfies the respective relation) due to two main reasons: first we want to use the idea of restriction in the *start/stop* configurations. In this way we make sure that some "artificial" constructs (for example encoding the whole RE set into the configurations) are not possible, and on the other hand, the regular languages express enough for the *start/stop* configurations to help in obtaining similar universality results as for "regular" systems with symport/antiport.

An important observation is the fact that we will not require the cell to "stop working" when reaching the result; i.e., we will not require the strong restriction that the system reaches a halting configuration for a computation to have a result. It is worth noting also that a similar idea of "configuration-based output" was considered recently in [7] where the authors pre-defined in the system two membranes: *fin* and *ack*; initially *ack* is empty, but once it receives an element, the number of objects hat are present in that moment in *fin* is the output of the computation. In this way one can consider the output of non-halting computations in that case as well. Of course, one can immediately see that in [7] the result is still encoded as multiplicities of different molecules, in our framework, we use the time between configurations to encode the result of the computation.

Before progressing any further we give some basic notions used in the remainder of the paper; the language theory notions used but not described are standard, and can be found in any of the many monographs available, for instance, in [15].

A membrane structure is pictorially represented by a Venn diagram, and it will be represented here by a string of matching parentheses.

A multiset over a set X is a mapping $M : X \longrightarrow \mathbf{N}$. Here we always use multisets over finite sets X (that is, X will be an alphabet). A multiset with a finite support can be represented by a string over X; the number of occurrences of a symbol $a \in X$ in a string $x \in X^*$, denoted by $|x|_a$, represents the multiplicity of a in the multiset represented by x. Clearly, all permutations of a string represent the same multiset, and the empty multiset is represented by the empty string, λ.

We will use symport rules of the form (ab, in) and (ab, out), associated with a membrane and stating that the objects a, b can enter, respectively, exit the membrane together, and antiport rules of the form $(a, out; b, in)$, stating that a exits and at the same time b enters the membrane.

Based on rules of these types, we modify the definition from [11] to introduce the model of a *timed symport/antiport P system* as the following construct:

$$\Pi = (O, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m, C_{start}, C_{stop}),$$

where

- $O = \{a_1, \ldots, a_k\}$ is an alphabet (its elements are called *objects*);
- μ is a membrane structure consisting of m membranes, with the membranes (and hence the regions) bijectively labeled with $1, 2, \ldots, m$; m is called the *degree* of Π;
- $w_i, 1 \leq i \leq m$, are strings over O representing multisets of objects associated with the regions $1, 2, \ldots, m$ of μ, present in the system at the beginning of a computation;
- $E \subseteq O$ is the set of objects that are continuously available in the environment in arbitrarily many copies;
- $R_1, \ldots, R_m$ are finite sets of symport and antiport rules over the alphabet V associated with the membranes $1, 2, \ldots, m$ of μ;
- C_{start} and C_{stop} are regular subsets of $(O^*)^m$, describing configurations of Π. We will use a regular language over $O \cup \{\$\}$ to describe them, the special symbol $\$ \notin O$ being used as a marker between the configurations in the different regions of the system. More details will be given in the following.

For a symport rule (x, in) or (x, out), we say that $|x|$ is the *weight* of the rule. The *weight* of an antiport rule $(x, out; y, in)$ is $\max\{|x|, |y|\}$. The rules from a set R_i are used with respect to membrane i as explained above. In the case of (x, in), the multiset of objects x enters the region defined by the membrane, from the surrounding region, which is the environment when the rule is associated with the skin membrane. In the case of (x, out), the objects specified by x are sent out of membrane i, into the surrounding region; in the case of the skin membrane, this is the environment. The use of a rule $(x, out; y, in)$ means expelling the objects specified by x from membrane i at the same time with bringing the objects specified by y into membrane i. The objects from E appear in arbitrarily many copies in the environment. The rules are used in the non-deterministic maximally parallel manner specific to P systems with symbol objects: in each step, a maximally parallel multiset of rules is used.

In this way, we obtain transitions between the configurations of the system. A configuration is described by the m-tuple of multisets of objects present in

the m regions of the system, as well as the multiset of objects from $O - E$ which were sent out of the system during the computation. It is important to note that such objects appear only in a finite number of copies in the initial configuration and can enter the system again (knowing the initial configuration and the current configuration in the membrane system, one can know precisely what "extra" objects are present in the environment). On the other hand, it is not necessary to take care of the objects from E which leave the system because they appear in arbitrarily many copies in the environment as defined before (the environment is supposed to be inexhaustible, irrespective how many copies of an object from E are introduced into the system, still arbitrarily many remain in the environment). The initial configuration is $\alpha_0 = (w_1, \ldots, w_m)$.

Let us now describe the way this systems "outputs" the result of its computation: when the system enters some configuration α from C_{start} (we also say that C_{start} is satisfied), we assume that an external observer (external to the cell) starts a counter t that is incremented at each clock cycle (or in other words, each time the symport/antiport rules are applied in the nondeterministic parallel manner). At some point, when the system enters some configuration β from C_{stop} (hence C_{stop} is satisfied), we stop incrementing t, and the value of t represents the output of the computation[1]. If the system never reaches a configuration in C_{start} or in C_{stop}, then we consider the computation unsuccessful, no output is associated with it. The set of all numbers t (computed as described above) is denoted by $N(\Pi)$. The family of all sets $N(\Pi)$ computed by systems Π of degree at most $m \geq 1$, using symport rules of weight at most p and antiport rules of weight at most q, is denoted by $NTP_m(sym_p, anti_q)$ (we use here similar notations as the ones from [11] and [2]).

We emphasize the fact that in the definition of Π we assume that C_{start} and C_{stop} are regular. Other, more restrictive, cases can be of interest but we do not discuss them here.

Details about P systems with symport/antiport rules can be found in [11]; a complete formalization of the syntax and the semantics of these systems is provided in the paper [13] where reachability of symport/antiport configurations was discussed.

3 Register Machines and Counter Automata

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

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

[1] By convention, in the case when a configuration α is reached that satisfies both C_{start} and C_{stop}, then we consider that the system has computed the value 0.

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

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

A register machine generates a natural number in the following manner: we start computing with all m registers being empty, with the instruction labeled by l_0; if the computation reaches the instruction $l_h : \texttt{HALT}$ (we say that it halts), then the values of register 1 is the number generated by the computation. The set of numbers computed by M in this way is denoted by $N(M)$.

We recall also the definition of the counter automaton; for more details we refer the interested to the literature [10]. Such a device is a construct $M = (m, Q, q_0, q_f, P)$, where d is the number of counters, $Q = \{q_0, \ldots, q_f\}$ is the set of states of the machine, q_0 is the start state while q_f is the final state and P is the set of instructions, of three types:

- $(p \to q, c+)$ with $p, q \in Q$ and c a counter. This instruction will increment the value of the register c and move from state p in state q.
- $(p \to q, c-)$ with $p, q \in Q$ and c a counter. The instruction tries to decrement the value of the counter c; if it was originally greater than zero, then it is decremented and M moves to the state q, otherwise (if the value stored in c is zero) the computation is stopped and the machine does not produce output.
- $(p \to q, c = 0)$ with $p, q \in Q$ and c a counter. The instruction tests the value of the counter c; if it is zero, then M moves to state q, otherwise the computation stops and the machine does not produce an output.

It is known (see [10]) that non-deterministic register machines and counter automata generate exactly the family NRE, of Turing computable sets of numbers.

4 Universality Results for Timed P Systems Having Only One Membrane

The first result that we give is related to the results obtained in [2] and [6] where it is proved that systems using only one membrane and symport rules of size 3 are universal. We need to mention that in our setup (by using the time as the output of the computation) we are able to generate all the subsets of NRE

including the ones containing the values from 0 to 7, which is not the case of the aforementioned results where there are some "garbage" symbols left in the output region. Another remark that should be made is that rather than using the more complicated notion of conflicting counters in a register machine, we use here a proof which is easier to understand and, implicitly, easier to implement.

Theorem 1. *P systems with time are universal for one membrane and symport of length 3 even when no antiport is used:* $NRE = NTP_1(sym_3, anti_0)$

Proof. We consider a register machine $M = (m, B, l_0, l_h, R)$ and we construct the system

$$\Pi = (O, [_1\ ]_1, w_1, E, R_1, C_{start}, C_{stop})$$

with the following components

$$O = \{a_r \mid 1 \leq r \leq m\} \cup \{P_l, P'_l, P''_l, Q_l, Q'_l, X_l, X'_l, X''_l, l \mid l \in B\} \cup \{b, t\},$$
$$w_1 = l_0 P_1 \dots P_l Q_1 \dots Q_l X_1 \dots X_l X''_1 \dots X''_l b^2,$$
$$E = \{a_r \mid 1 \leq r \leq m\} \cup \{P'_l, P''_l, Q'_l, X'_l, l \mid l \in B\} \cup \{t\},$$
$C_{start} = \{b^2 t^2 w_1 \mid w_1 \in (O - \{b, t\})^*\}$, in other words, b and t appear exactly two times, and the rest of the symbols can appear in any numbers.

$C_{stop} = \{t^i w_2 \mid i \geq 1, w_2 \in (O - \{a_1\})^*\}$, in this case we have that t appears at least once, while a_1 does not appear in the region.

and the following rules in R_1:

1. For an ADD instruction $l_1 : (\texttt{ADD}(r), l_2, l_3) \in R$, we consider the rules

   $(P_{l_1} l_1, out),\ (P_{l_1} l_2 a_r, in),\ (P_{l_1} l_3 a_r, in).$

   We simulate the work of the ADD instruction in two steps. First we send out the current instruction label together with the marker P_{l_1} that will come back in the membrane with two other objects, a copy of a_r so that the register r is incremented and also the new instruction label. To simulate the non-deterministic behavior of these machines we have two symport rules that do the same job, the only difference being the next instruction label being brought back in the system. It is clear that the simulation of the ADD instruction is performed correctly.
2. For a SUB instruction $l_1 : (\texttt{SUB}(r), l_2, l_3) \in R$ we consider the following rules:

   $(P_{l_1} l_1, out),\ (P_{l_1} P'_{l_1} P''_{l_1}, in),\ (P'_{l_1} Q_{l_1}, out),\ (P''_{l_1} X_{l_1} a_r, out),\ (Q_{l_1} Q'_{l_1}, in),$
   $(X_{l_1} X'_{l_1} l_2, in),\ (Q'_{l_1} P''_{l_1} X''_{l_1}, out),\ (Q'_{l_1} X'_{l_1}, out),\ (X''_{l_1} l_3, in).$

   We simulate the work of the SUB instruction in several steps (5 if the register is not empty and 6 if it is empty). We first send out the current label with its corresponding P marker by the rule $(P_{l_1} l_1, out)$. At the next step the symbol P brings in two more symbols that keep track of the instruction being simulated with theirs indices: $(P_{l_1} P'_{l_1} P''_{l_1}, in)$, P' is working as a timer while P'' is checking whether the register is empty or not. If the register is

not empty, then P'' will exit decreasing the register and taking at the same time another marker to the outside to help identify the correct case later: $(P''_{l_1} X_{l_1} a_r, out)$. At the next stage X will return with yet another marker and the next instruction label to be brought in (in this case l_2 as the register was not empty), $(X_{l_1} X'_{l_1} l_2, in)$. The work is finished in this case by the rule $(Q'_{l_1} X'_{l_1}, out)$.

If the register is empty, we perform the same initial steps, sending P and the current instruction label out, P returns with P' and P''; this time P'' cannot exit the membrane at the next step since the register is empty, but P' is exiting together with Q, then Q returns with Q'. At the next step we have the "branching point": rather than exiting with X' (which will be present in the membrane in the case when the register was not empty), Q' exits with P'' and X''. If X'' exits, that means that the register was empty, thus when X'' returns in the system, it returns with the label of the next instruction to be simulated as l_3.

3. The terminating/counting work is done by the rules:

   $(l_h b^2, out),\ (bt, in),\ (bta_1, out).$

   It is clear that at the end of the simulation, if the register machine has reached the final state, we will also have the halting instruction symbol in the system membrane. At that time we will have the computed value encoded as the multiplicity of the object a_1 that is associated with the output register. We will also have in the system the label of the halting instruction, l_h, thus the rule $(l_h b^2, out)$ can be applied only when the simulation was performed correctly. At the next step, the two b-s return with two copies of t, satisfying the C_{start} configuration. One can note that if there are no copies of a_1 in the membrane, then also the configuration C_{stop} is satisfied at the same time, thus our system would compute the value 0 in that case. For any even value encoded in the multiplicity of a_1 it will take exactly half that number of steps for the two copies of the pair bt to push the a_1-s out of the membrane and again the same amount to return to the membrane.

   Let us give a small example: for the value 4, the first step 2 copies of a_1 are pushed out, and at the next step the symbols b^2t^2 return in the membrane; in two more steps we will have 0 copies of a_1 and at least one copy of t, so the whole process took the correct 4 steps to complete.

   For an odd number we perform the same work, with the exception of the last step, when there is only one copy of a_1 in the membrane. At that moment, only one bt can exit, leaving the second one in the membrane, satisfying the C_{stop} condition at the correct time. □

In the following we recall a proof from [8] due to its relevance to the current paper. The best known result for "standard" symport/antiport P systems in this setting is $N_1RE = N_1OP_1(sym_0, anti_2)$, thus the following result improves the best known result for symport/antiport systems by being able to generate sets of numbers containing also the values 0 and 1. Another observation is that the $C_{start},\ C_{stop}$ configuration are in this case quite simple. We call them having

minimal restrictions on multiplicities; we mean by this the fact that for each object and each membrane, the C_{start}, C_{stop} rules will impose either a fixed multiplicity or not impose any restrictions for the object.

Theorem 2. *Using minimal restrictions on the multiplicities of the objects for the C_{start}, C_{stop} rules we have $NRE = NTP_1(sym_0, anti_2)$.*

Proof. To prove the theorem we follow the constructions from the literature for the "standard" symport/antiport P systems, this time using counter automata. In the initial configuration, the unique membrane contains the start state as its only object. The work of the counter automaton can be simulated using the antiport rules in the following way.

For a rule $(p \to q, \lambda) \in R$ we will have in our timed P system the rule $(q, in; p, out)$; for an increment instruction $(p \to q, i+)$ on the counter c_i we will add the antiport rule $(qc_i, in; p, out)$ to R_1. The decrement instruction can only be applied if the counter is non zero: $(p \to q, i-)$ is simulated by $(q, in; pc_i, out)$. Finally, $(p \to q, i = 0)$ is simulated by the rules $(q'\overline{i}, in; p, out)$; $(\infty, in; \overline{i}c_i, out)$, $(q'', in; q', out)$, and $(q, in; q''\overline{i}, out)$ in three steps: first we replace p by q' and $\overline{i}$, then $\overline{i}$ checks whether the register i is empty or not; if nonempty, the special marker ∞ is brought in and the computation cannot continue; in the case when the register was empty the computation can continue by expelling the two symbols q'' and $\overline{i}$ together to bring in the next state q.

It is clear now that the register machine is simulated in this way only by using antiport rules of weight[2] 2. When the final state appears as the current state of the simulation it is time to start "counting" the result. We define $C_{start} = \{wf \mid w \in (O - \{f, \infty\})^*\}$. The rule $(f, in; fc_0, out)$ will expel one symbol c_0 at a time, thus if we define $C_{stop} = \{fw' \mid w' \in (O - \{f, c_0\})^*\}$, we will have exactly i steps between C_{start} and C_{stop}, where i is the multiplicity of the symbol c_0 (i.e., the contents of the output register) in the system. Following the previous discussion the equality $NRE = NTP_1(sym_0, anti_2)$ was shown, which completes the proof. □

5 Universality Results for Timed P Systems Having Two Membranes

In this section we will provide two dual results with the ones presented in [1], [2] when considering systems with two membranes. It is worth noting that in [1] the authors use and intersection with a finite alphabet when defining the result of a computation. The best result from [2] is $N_3RE = N_3OP_2(sym_1, anti_1)$. Here we improve this result in the sense that we generate also the sets of numbers containing the values 0 through 3. We will give in the following theorem the

[2] The result can be strengthened in the following way: the construction works even if we only use antiport rules of dimensions $(1, 2)$ or $(2, 1)$ by adding to the only two rules of dimension $(1, 1)$ some padding symbols. For example the rule $(q'', in; q', out)$ can be padded with the extra symbol P in this way $(q''P, in; q', out)$.

C_{start}/C_{stop} configurations in the form <multiset for membrane 1>$<multiset for membrane 2> described by regular languages as defined in the second section of the paper; in the first two proofs we showed universality of a single region, thus the symbol $ was not used.

Theorem 3. $NRE = NTP_2(sym_1, anti_1)$.

Proof. We will follow the construction from [1] and note the changes made. For a detailed explanation of the work of the system we refer the interested reader to [1].

Let us consider a counter automaton $M = (m, Q, q_0, q_f, P)$ which starts with empty counters and has n instructions in the set P.

We construct the P system $\Pi = (O, [_1[_2\ ]_2]_1, w_1, w_2, E, R_1, R_2, C_{start}, C_{stop})$ with the following components

$$\begin{aligned}
O &= E \cup \{b_j, b'_j \mid 1 \le j \le n\} \cup \{\#, F, I\},\\
w_1 &= q_0 IF\#\#,\\
w_2 &= b_1 b_2 \dots b_n b'_1 b'_2 \dots b'_n dd,\\
E &= Q \cup \{c_r \mid 1 \le r \le m\} \cup \{a_j, a'_j, a''_j \mid 1 \le j \le n\} \cup \{z\},\\
C_{start} &= \{d\#^2 w_1 \$ w_2 \mid w_1 \in (O - \{d, \#\})^*, w_2 \in (O - \{d\})^*\},\\
C_{stop} &= \{z w_3 \$ w_4 \mid w_3 \in O^*, w_4 \in (O - \{c_1\})^*\}.
\end{aligned}$$

The definition of C_{start} means that for starting to count we need to reach a state when exactly one copy of d and two copies of # are present in membrane 1, at the same time as no copy of d is present in membrane 2. At the same time, C_{stop} is only satisfied when at least one copy of the symbol z is present in membrane 1 and no copies of c_1 appears in membrane 2.

Let us now define the rules from R_1 and R_2:

$$\begin{aligned}
R_1 &= R_1^{ini} \cup R_1^{sim} \cup R_1^{timer}, \text{ and}\\
R_2 &= R_2^{ini} \cup R_2^{sim} \cup R_2^{timer}, \text{ where}\\
R_1^{ini} &= \{(I, out; c_k, in) \mid 1 \le k \le m\} \cup \{(I, in)\},\\
R_2^{ini} &= \emptyset,\\
R_1^{sim} &= \{(q_i, out; a_j, in), (a''_j, out; q_l, in) \mid (j : q_i \to q_l, .) \in P\}\\
&\quad \cup \{(b_j, out; a'_j, in), (a_j, out; b_j, in), (\#, out; b_j, in) \mid 1 \le j \le n\}\\
&\quad \cup \{(a'_j, out; a''_j, in) \mid \text{ where } j \text{ is the label of an increment or decrement}\\
&\qquad \text{instruction}\} \cup \{(\#, out; \#, in)\}\\
&\quad \cup \{(b'_j, out; a''_j, in), (a'_j, out; b'_j, in) \mid (j : q_i \to q_l, k = 0) \in P\}\\
&\quad \cup \{(\#, out; b'_j, in) \mid (j : q_i \to q_l, k = 0) \in P\}\\
R_2^{sim} &= \{(b_j, out; a_j, in) \mid 1 \le j \le n\}\\
&\quad \cup \{(a_j, out; c_k, in), (a'_j, in) \mid (j : q_i \to q_l, k+) \in P\}\\
&\quad \cup \{(a'_j, out; b_j, in) \mid j \text{ labels an increment or decrement instruction}\}\\
&\quad \cup \{(a_j, out) \mid j \text{ labels a decrement or test with 0 instruction}\}
\end{aligned}$$

$$\cup \{(c_k, out; a'_j, in) \mid (j : q_i \to q_l, .) \in P\}$$
$$\cup \{(b'_j, out; b_j, in), (b'_j, in) \mid (j : q_i \to q_l, k = 0) \in P\},$$
$$R_1^{timer} = \{(d, out; z, in)\},$$
$$R_2^{timer} = \{(d, out; q_f, in), (q_f, out; F, in), (c_1, out; z, in), (z, out)\}.$$

The simulation of the counter automaton is done in phases. The rules from the *ini* phase bring in membrane 1 an arbitrary number of objects c_i for any register i. This helps with the simulation of the increment instruction in the automaton. The rules in the *sim* group perform the actual simulation of the different instructions in the automaton (increment, decrement, test for 0). We refer the interested reader to [1] for the details of the construction and the proof of correctness. We change the construction in the finishing part of the simulation to match the output based on the time that passes between two configurations. For this we note that at the start of each simulation step we have a symbol codifying the current state in the automaton q_i in membrane 1, whereas membrane 2 holds the current values of the counters codified by multiplicities of the symbols c_j.

After a successful simulation, we will reach a state when q_f is in membrane 1 and some number of c_1 objects that are present in membrane 2 codify the output of the system. At this moment we can use the rules from the set *timer*. We will give step-by-step explanations for this phase.

At step 1 q_f is in membrane 1 and dd in membrane 2. By using the rule $(d, out; q_f, in)$ we will now have d in membrane 1 and dq_f in membrane 2. At the next step d from membrane 1 is replaced by a z using the rule $(d, out; z, in)$, and at the same time q_f returns in membrane 1 by the rule $(q_f, out; F, in)$. At the next step q_f will remove the second d from membrane 2, thus satisfying the C_{start} condition. There are two cases: a) the output register was empty and b) the output register is not empty.

In the case a) we have the following configuration: in membrane 1 we have zd and in membrane 2 we have $q_f F$ and no copies of c_1; this configuration that satisfied C_{start} will satisfy also C_{stop}, thus the value computed is correctly reported as 0.

Case b) if there was at least on copy of c_1 in membrane 2, then the rule $(c_1, out; z, in)$ is applicable, so now membrane 1 has d and membrane 2 has $zq_f F$, and this satisfies C_{start}. At the next moment the symbol d will be replaced by a second z in membrane 1, while the first z returns to membrane 1 by the rule (z, out). If the value of the output counter was 1, then there are no more copies of c_1 in membrane 2, thus the configuration satisfies at this step C_{stop}, thus correctly computing the value 1. If the register stored a value more than 2, then the computation continues in a homogenous fashion from now on: the two copies of z will expel each a copy of the c_1 marker and at the next step return to membrane 1. If the amount of objects c_1 was odd, then the computation finishes with the two symbols z in membrane 1 and no symbols c_1 in membrane 2, thus reaching the C_{stop} in a correct amount of time. On the other hand, if the output of the computation was an even value, then one of the z symbols will be swapped

with the last c_1, while the second z will remain in membrane 1, making the C_{stop} satisfiable, thus computing also in this case the correct number of steps. This concludes the proof. □

We will now proceed to prove the last result of the paper which still deals with universality of timed symport/antiport P systems. We prove that two membranes and symport of size 2 are enough for generating all the NRE sets. The best result for the case of symport/antiport systems with output in an elementary membrane is given in [2] where the authors show that such systems with two membranes and symport of size 2 are universal, but cannot generate sets of numbers containing the values 0 through 6. In the case of systems based on time, we match the universality result of systems with two membranes and symport of size two, but we are also able to generate the sets of numbers containing the values 0 through 6.

Theorem 4. $NRE = NTP_2(sym_2, anti_0)$.

Proof. We will follow again the construction from [1] and note the changes made to it.

Let us consider as in the proof of Theorem 3 a counter automaton $M = (m, Q, q_0, q_f, P)$ which starts with empty counters and has n instructions. We construct the P system $\Pi = (O, [_1[_2\ ]_2]_1, w_1, w_2, E, R_1, R_2, C_{start}, C_{stop})$ with the following components:

$$
\begin{aligned}
O &= E \cup Q \cup \{b_j, g_j \mid 1 \le j \le n\} \cup \{g'_j \mid 1 \le j \le n-1\} \cup \{\#, \$, F, y\},\\
w_1 &= q_0 a_1 F \$ b_1 b_2 \dots b_n,\\
w_2 &= \#q_1 q_2 \dots q_f g_1 g_2 \dots g_n g'_1 g'_2 \dots g'_{n-1} yy,\\
E &= \{c_r \mid 1 \le r \le m\} \cup \{a_j, a'_j, d_j, d'_j \mid 1 \le j \le n\},\\
C_{start} &= \{y^2 w_1 \$ \# w_2 \mid w_1 \in (O - \{y\})^*, w_2 \in (O - \{\#\})^*\},\\
C_{stop} &= \{y w_3 \$ w_4 \mid w_3 \in (O - \{c_1\})^*, w_4 \in O^*\}.
\end{aligned}
$$

The rules from R_1 and R_2 are as follows:

$$
\begin{aligned}
R_1 &= R_1^{sim}, \text{ and}\\
R_2 &= R_2^{sim} \cup R_2^{timer}, \text{ where}\\
R_1^{sim} &= \{(q_i a_j, in) \mid (j : q_i \to q_l, .) \in P\}\\
&\quad \cup \{(b_j g_j, out) \mid j \text{ is the label of a increment or zero check}\\
&\quad \cup \{(c_k b_j, in) \mid (j : q_i \to q_l, k+) \in P,\ q_i, q_l \in Q,\ 1 \le k \le m\}\\
&\quad \cup \{(c_k g_j, out) \mid (j : q_i \to q_l, k-) \in P,\ q_i, q_l \in Q,\ 1 \le k \le m\}\\
&\quad \cup \{(a'_j g_j, in) \mid 1 \le j \le n-1\} \cup \{(\#, out), (\#, in)\}\\
&\quad \cup \{(d_j b_j, in), (d_j c_k, out), (a'_j g'_j, out) \mid (j : p \to q, k = 0) \in P\}\\
&\quad \cup \{(d'_j g'_j, in), (d'_j, out) \mid (j : p \to q, k = 0) \in P\}
\end{aligned}
$$

$$\cup \{(a'_j q_l, out) \mid (j : q_i \to q_l, k+) \in P \text{ or } (j : q_i \to q_l, k-) \in P\}$$
$$\cup \{(d_j q_l, out) \mid (j : q_i \to q_l, k = 0) \in P,\ 1 \le k \le m\},$$
$$R_2^{sim} = \{(a_j b_j, in), (b_j g_j, out), (a'_j g_j, in) \mid 1 \le j \le n-1,\ (j : q_i \to q_l, .) \in P\}$$
$$\cup \{(q_i, in) \mid q_i \in Q\} \cup \{(a'_j \$, in) \mid 1 \le j \le n\} \cup \{(\#\$, out)\}$$
$$\cup \{(a'_j q_l, out) \mid (j : q_i \to q_l, k+) \in P \text{ or } (j : q_i \to q_l, k-) \in P\}$$
$$\cup \{(a'_j g'_j, out), (d'_j g'_j, in), (d_j q_l, out) \mid$$
$$(j : q_i \to q_l, k = 0) \in P,\ 1 \le k \le m\}$$
$$R_2^{timer} = \{(q_f F, in), (q_f y, out), (Fy, out), (yc_1, in)\}.$$

The simulation of the register machine is done by the rules in the *sim* groups. We refer the interested reader to [1] for the details of the construction and the proof of correctness. We note that the construction from [1] was changed for our purposes, thus the rules from the group *start* as they were defined in the Theorem 2 in [1] are no longer needed since we do not need the "safety net" of the computation running forever (done by using the symbol $\#_2$) if the simulation is blocked. This is due to the flexibility of the C_{start}/C_{stop} configurations. We note that the construction is simulating the work of the counter automaton in membrane 1 and use membrane 2 as a filtering mechanism. Membrane 1 contains both the values of the counters (codified as multiplicities of objects c_i), the current state of the automaton (codified as an object q_j) and the next instruction to be executed (codified as a symbol a_k). At the end of a successful simulation we will have q_f in membrane 1, some number of objects c_1 codifying the output of the simulation and # in membrane 2 (if # has reached membrane 1, then the simulation was not correct). The rules in the *timer* group can only be applied after the simulation was completed successfully, thus the symbol q_f appears in membrane 1. At that time q_f together with F enter membrane 2 by rule $(q_f F, in) \in R_2^{timer}$, where they are able to "pair" with a copy of y each and exit to membrane 1, by using the rules $(q_f y, out), (Fy, out) \in R_2^{timer}$. At this moment we will have again q_f and F in membrane 1, and also y^2 in the same membrane. It is easy to see that the C_{start} is satisfied now since the two carrier symbols y are in membrane 1. If the value in register 1 is zero, then the C_{stop} is also valid producing the correct output for this case. If the value computed by the automaton is non-zero, then the objects y will start moving copies of c_1 in membrane 2, and return to membrane 1 using the symbols q_f and F. This is done up until all the symbols c_1 have been moved in region 2. We have two cases for this process: a) the number computed by the automaton is even or b) the number computed is odd.

In the case a) one can notice that it takes two steps for a symbol y to return to membrane 1, but since there are exactly two such symbols in the system, for every two steps of the system, two c_1-s are moved to region 2 and the two y symbols return to membrane 1. Thus in the same amount of steps as the number computed, the configuration C_{stop} becomes satisfied – to this aim, it needs at least one copy of y present in membrane 1, so we need in this case to wait until both y-s return to membrane 1 after the c_1-s are depleted. In the case b) one

can note that in some even number of steps, $2s$ for example, exactly $2s$ copies of c_1 are moved to region 2 and the y-s return to membrane 1, thus without loss of generality we can assume that in $2s$ steps we only have one more copy of c_1 in membrane 1. In that moment, one of the y-s will move the c_1 to region 2, while the other copy of y cannot move, thus at the next step we reach a configuration from C_{stop}, and the system outputs correctly the value $2s + 1$. □

6 Final Remarks

For the newly introduced timed P systems we improved or matched the four best known results for "regular" symport/antiport P systems. It is worth noting that the new feature of outputting the result using time is more flexible than the previously considered methods, thus the previous results could be even improved by using completely different techniques that take advantage of the flexibility of the time as a framework of outputting the result of a computation. For example in the new framework we no longer have the (rather strong) requirement that the computation should halt, only to reach a configuration from C_{start} and then one from C_{stop}. We conjecture that this new definition could prove useful also in conjunction with classes of symport/antiport systems that using the original definition could only generate finite sets (e.g., generate some non-finite family of numbers, etc.).

Acknowledgments

A. Păun gratefully acknowledges the support in part by LA BoR RSC grant LEQSF (2004-07)-RD-A-23 and NSF Grants IMR-0414903 and CCF-0523572.

References

1. A. Alhazov, R. Freund, Yu. Rogozhin, Some Optimal Results on Symport/Antiport P Systems with Minimal Cooperation, M.A. Gutiérrez-Naranjo et al. (eds.), Cellular Computing (Complexity Aspects), ESF PESC Exploratory Workshop, Fénix Editora, Sevilla (2005), 23–36.
2. A. Alhazov, R. Freund, Yu. Rogozhin, Computational Power of Symport/Antiport: History, Advances and Open Problems, R. Freund et al. (eds.), Membrane Computing, International Workshop, WMC 2005, Vienna (2005), revised papers, *Lecture Notes in Computer Science* 3850, Springer (2006), 1–30.
3. F. Bernardini, A. Păun, Universality of Minimal Symport/Antiport: Five Membranes Suffice, WMC03 revised papers in *Lecture Notes in Computer Science* 2933, Springer (2004), 43–54.
4. M. Cavaliere, R. Freund, Gh. Păun, Event–Related Outputs of Computations in P Systems, M.A. Gutiérrez-Naranjo et al. (eds.), Cellular Computing (Complexity Aspects), ESF PESC Exploratory Workshop, Fénix Editora, Sevilla (2005), 107–122.

5. R. Freund, A. Păun, Membrane Systems with Symport/Antiport: Universality Results, in *Membrane Computing. Intern. Workshop WMC-CdeA2002, Revised Papers* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), *Lecture Notes in Computer Science*, 2597, Springer-Verlag, Berlin (2003), 270–287.
6. P. Frisco, J.H. Hogeboom, P systems with Symport/Antiport Simulating Counter Automata, *Acta Informatica*, 41 (2004), 145–170.
7. P. Frisco, S. Ji, Towards a hierarchy of conformon-P systems, *Membrane Computing. International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 2002, Revised Papers* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), Springer, Berlin, 2003, 302–318,
8. O.H. Ibarra, A. Păun, Counting Time in Computing with Cells, Proceedings of DNA11 conference, June 6-9, 2005, London Ontario, Canada, (14 pages).
9. M. Ionescu, Gh. Păun, T. Yokomori, Spiking Neural P Systems, *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
10. M.L. Minsky, Recursive Unsolvability of Post's Problem of "Tag" and Other Topics in Theory of Turing Machines, *Annals of Mathematics*, 74 (1961), 437–455.
11. A. Păun, Gh. Păun, The Power of Communication: P Systems with Symport/Antiport, *New Generation Computing*, 20, 3 (2002) 295–306.
12. Gh. Păun, Further Twenty-six Open Problems in Membrane Computing, the Third Brainstorming Meeting on Membrane Computing, Sevilla, Spain, February 2005.
13. Gh. Păun, M.J. Pérez-Jiménez, F. Sancho-Caparrini, On the Reachability Problem for P Systems with Symport/Antiport, *Proc. Automata and Formal Languages Conf.*, Debrecen, Hungary, 2002.
14. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, Spike Trains in Spiking Neural P Systems, *International Journal of Foundations of Computer Science*, in press.
15. G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Berlin, 1997.

Towards Probabilistic Model Checking on P Systems Using PRISM

Francisco J. Romero-Campero[1], Marian Gheorghe[2], Luca Bianco[3], Dario Pescini[4], Mario J. Pérez-Jiménez[1], and Rodica Ceterchi[5]

[1] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville, Avda. Reina Mercedes, 41012 Sevilla, Spain
{fran,marper}@cs.us.es
[2] Department of Computer Science, The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
M.Gheorghe@dcs.shef.ac.uk
[3] Department of Computer Science, University of Verona
Strada Le Grazie 15, 37134 Verona, Italy
bianco@sci.univr.it
[4] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
pescini@disco.unimib.it
[5] University of Bucharest, Faculty of Mathematics and Computer Science
Academiei 14, 70109 Bucharest, Romania
rc@funinf.cs.unibuc.ro

Abstract. This paper presents the use of P systems and π-calculus to model interacting molecular entities and how they are translated into a probabilistic and symbolic model checker called PRISM.

1 Introduction

The complexity of bio-molecular cell systems is currently the focus of intensive experimental research, nevertheless the enormous amount of data about the function, activity, and interactions of such systems makes necessary the development of models able to provide a better understanding of the dynamics and properties of the systems. A model, an abstraction of the real-world onto a mathematical/computational domain, highlights some key features while ignoring others that are assumed to be not relevant. A good model should have (at least the following) four properties: relevance, computability, understandability and extensibility, [20]. A model must be relevant capturing the essential properties of the phenomenon investigated; and computable so it can allow the simulation of its dynamic behavior, and the qualitative and quantitative reasoning about its properties. An understandable model will correspond well to the informal concepts and ideas of molecular biology. Finally, a good model should be extensible to higher levels of organizations, like tissues, organs, organisms, etc., in which molecular systems play a key role.

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 477–495, 2006.

In this paper we will deal with models developed within the framework of membrane computing. Membrane computing is an emergent branch of natural computing introduced by G. Păun in [15]. This new model of computation starts from the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations. The devices of this model are called P systems. Roughly speaking, a P system consists of a cell-like membrane structure, in the compartments of which one places multisets of objects which evolve according to given rules.

Although most research in P systems concentrates on the computational power of the devices involved, lately they have been used to model biological phenomena within the framework of computational systems biology. In this case P systems are not used as a computing paradigm, but rather as a formalism for describing the behavior of the system to be modeled. In this respect several P systems models have been proposed to describe oscillatory systems [8], signal transduction [17], gene regulation control [16], quorum sensing [12,18,21] and metapopulations [19]. These models differ in the type of the rewriting rules, membrane structure and the strategy applied to run the rules in the compartments defined by membranes. Some of these models using the *metabolic algorithm* [5], the *dynamical probabilistic P systems* [19], and *(multicompartmental) Gillespie Algorithm* [17] were applied in certain case studies.

As P systems are inspired from the structure and functioning of the living cell, it is natural to consider them as modeling tools for biological systems, within the framework of systems biology, being an alternative to more classical approaches like ordinary differential equations (ODEs) and to some recent approaches like Petri nets and π-calculus. Differential equations have been used successfully to model kinetics of conventional macroscopic chemical reactions where the main focus is on the average evolution of the concentration of chemical substances across the whole system. Nevertheless, there is an implicit assumption of continuously varying chemical concentration and deterministic dynamics. Two critical characteristics of this approach are that the number of molecules of each type in the reaction mix is large and that for each type of reaction in the system, the number of reactions is large within each observation interval, that is reactions are fast.

When the number of particles of the reacting species is small and reactions are slow, which is frequently the case in some biological systems, both previous assumptions are questionable and the deterministic continuous approach to chemical kinetics should be complemented by an alternative approach. In this respect, one has to recognize that the individual chemical reaction steps occur discretely and are separated by time intervals of random length. Stochastic and discrete approaches like the ones used with Petri nets [11], π-calculus [20] and P systems [17,19] are more accurate in this situation. Nevertheless, these formalisms differ in some essential features that will be discussed briefly in this paper.

Most research in systems biology focuses on the development of models of different biological systems in order to be able to simulate them, accurately enough such as to be able to reveal new properties that can be difficult or impossible to discover through direct experiments. One key question is what one can do with a model, other than just simulate trajectories? This question has been considered in detail for deterministic models, but less for stochastic models. Stochastic systems defy conventional intuition and consequently are harder to conceive. The field is widely open for theoretical advances that help us to reason about systems in a greater detail and with a finer precision.

An attempt in this direction consists in using model checking tools to analyze in an automatic way various properties of the model. There are previous studies investigating the use of model checking for P system specifications [2,7].

Our current attempt uses a probabilistic symbolic model checking approach based on PRISM (Probabilistic and Symbolic Model Checker) [22] and investigates continuous time P systems with Gillespie dynamics using protein-protein interaction rules.

Systems consisting of interacting molecular entities have been modeled by using π-calculus formalism [20] explaining the principles of transforming the biological system into a π-calculus model in a coherent way.

In this paper it is shown how π-calculus and P systems can model systems consisting of reactions with biochemical entities. The specification is translated into PRISM and various properties are studied. Some simulations obtained using the PRISM simulator as well as a P system simulator with Gillespie dynamics are presented.

The paper is organized as follows: in Section 2 a brief overview of PRISM is presented; Section 3 deals with P system specifications in PRISM, Section 4 presents a case study representing the cell cycle in eukaryotes described using a P system specification and a π-calculus definition; both are then translated into PRISM and contrasted in Section 5; conclusions are drawn in Section 6.

2 PRISM

Probabilistic model checking is a formal verification technique. It is based on the construction of a precise mathematical model of a system which is to be analyzed. Properties of this system are then expressed formally using temporal logic and analyzed against the constructed model by a probabilistic model checker.

PRISM, the probabilistic and symbolic model checker in this study, supports three different types of probabilistic models, discrete time Markov chains (DTMC), Markov decision processes (MDP), and continuous time Markov chains (CTMC). PRISM supports systems specifications through two temporal logics, PCTL (probabilistic computation tree logic) for DTMC and MDP and CSL (continuous stochastic logic) for CTMC.

In order to construct and analyze a model with PRISM, it must be specified in the PRISM language, a simple, high level, state-based language based on the Reactive Modules formalism of [1].

Here we describe some aspects of the PRISM language through the following illustrative example taken from [22].

```
// N-place queue + server

ctmc

const int N = 10;
const double mu = 1/10;
const double lambda = 1/2;
const double gamma = 1/3;

module queue
      q : [0 .. N] init 0;

      [] q < N -> mu : (q' = q + 1);
      [] q = N -> mu : (q' = q);
      [serve] q > 0 -> lambda : (q' = q - 1);

endmodule

module server
      s : [0 .. 1] init 0;

      [serve] s = 0 -> 1 : (s' = 1 );
      [] s = 1 -> gamma : (s' = 0);

endmodule
```

The fundamental components of the PRISM language are modules, variables and commands. A model is composed of a number of modules which can interact with each other. A module contains a number of local variables and commands.

The previous example consists of two modules; the first one represents a `queue` and the second one represents a `server`.

A module is specified as:

```
module ⟨name⟩

endmodule
```

Note that, in the example above, there are only two local variables, `q` in the `queue` module representing the size of the queue, and `s` in the `server` module which represents whether or not the server is busy. In the declaration of a variable its initial value and range must be specified. A variable declaration looks like:

```
name : [ lower-bound .. upper-bound ] init value;
```

The values of these variables at any given time constitute the states of the module. The space of reachable states is computed using the range of each variable and its initial value. The global state of the whole model is determined by the local state of all modules.

The behavior of each module is described by a set of commands. A command takes the form:

$$\texttt{[action]}\ \mathtt{g} \rightarrow \lambda_{\mathbf{1}} : \mathbf{u_1} + \cdots + \lambda_{\mathbf{n}} : \mathbf{u_n};$$

The guard `g` is a predicate over all the variables of the model. Each update $\mathbf{u_i}$ describes the new values of the variables in the module specifying a transition of the module. The expressions $\lambda_{\mathbf{i}}$ are used to assign probabilistic information, rates, to transitions.

The label `action` placed inside the square brackets are used to synchronize the application of different commands in different modules. This forces two or more modules to make transitions simultaneously. The rate of this transition is equal to the product of the individual rates, since the processes are assumed to be independent.

In our example, in the `queue` module there are three commands; the first one allows a new client to join the queue with probability `mu` if the maximal size, `N`, has not been reached yet; otherwise the second command maintains the size of the queue constant with probability `mu`. The third command is synchronized with the first command of the `server` module and describes the situation when there are clients in the queue and the server is free; in this case with rate `lambda` the server is set to busy and one client is removed from the queue. Observe that the rate of this transition is equal to the product of the two individual rates ($1 \times$ `lambda` = `lambda`), this is a common technique, an action is *passive* with rate 1 and the other action *active* which actually defines the rate for the synchronized transition.

PRISM supports many other features like constants, expressions, process algebra operators, etc. For a detailed description of the tool we refer to [22].

3 Transforming P System Specification into PRISM

The main components of a P system are a membrane structure consisting of a number of membranes that can interact with each other, an alphabet of objects, and a set of rules associated to each membrane. These components can easily be mapped into the components of the PRISM language using modules to represent membranes, variables to describe the alphabet and commands to specify the rules.

A P system is a construct

$$\Pi = (\Sigma, L, \mu, M_1, M_2, \ldots, M_n, R_1, \ldots, R_n)$$

where:

- Σ is a finite alphabet of symbols representing objects;
- L is a finite alphabet of symbols representing labels for the compartments[1];
- μ is a membrane structure containing $n \geq 1$ membranes labeled with elements from L;
- $M_i = (l_i, w_i)$, for each $1 \leq i \leq n$, is the initial configuration of membrane i with $l_i \in L$ and $w_i \in \Sigma^*$ a finite multiset of objects;
- R_i, for each $1 \leq i \leq n$, is a finite set of rules in membrane i of the form specified below with objects in Σ and labels in L.

The types of rules we will consider in this paper are those referred in the literature as protein-protein interaction rules.

- Transformation, complex formation and dissociation rules:

$$[\ a\ ]_l \xrightarrow{c} [\ b\ ]_l$$

$$[\ a,\ b\ ]_l \xrightarrow{c} [\ e\ ]_l \quad \text{where } a, b, e \in \Sigma \text{ and } l \in L$$

$$[\ a\ ]_l \xrightarrow{c} [\ b,\ e\ ]_l$$

  These rules are used to specify chemical reactions taking place inside a compartment of type $l \in L$, more specifically they represent the transformation of a into b, the formation of a complex e from the interaction of a and b, and the dissociation of a complex a into b and e respectively. These types of rules are used for example in [5] to describe oscillations as a consequence of the interactions between different objects inside a single compartment.

- Diffusing in and out:

$$\begin{array}{l} [\ a\ ]_l \xrightarrow{c} a\ [\ ]_l \\ \qquad\qquad\qquad \text{where } a \in \Sigma \text{ and } l \in L \\ a\ [\ ]_l \xrightarrow{c} [\ a\ ]_l \end{array}$$

  When chemical substances move or diffuse freely from one compartment to another one we use these types of rules, where a moves from or to a compartment of type l.

  These rules are also used to model metapopulations [19] where individuals can move from one compartment to another one or signal molecules occurring in bacteria [17] using population P systems [4] as a model.

[1] Two membranes with the same label will be considered of the same *type.*

– Binding and debinding rules:

$$a\,[\,b\,]_l \xrightarrow{c} [\,e\,]_l$$
$$[\,a\,]_l \xrightarrow{c} b\,[\,e\,]_l$$
where $a, b, e \in \Sigma$ and $l \in L$

Using rules of the first type we can specify reactions expressing the binding of a ligand swimming in one compartment to a receptor placed on the membrane surface of another compartment. The reverse reaction, debinding of a substance from a receptor, can also be described by using the second rule. These rules were used to model signalling at the cell surface in [17].

– Recruitment and releasing rules:

$$a\,[\,b\,]_l \xrightarrow{c} e\,[\;\;]_l$$
$$e\,[\;\;]_l \xrightarrow{c} a\,[\,b\,]_l$$
where $a, b, e \in \Sigma$ and $l \in L$

With these rules we represent the interaction between two chemicals in different compartments whereby one of them is recruited from its compartment by a chemical on the other compartment, and then the new complex remains in the latter compartment. In a releasing rule a complex, e, located in one compartment can dissociate into a and b, with a remaining in the same compartment as e, and b being released into the other compartment. In [17], these rules were used to describe the signal transduction between environmental concentrations of signal molecules and the cytoplasmic concentrations of different kinases.

Here, in order to capture the features of all these rules, we consider generic rules of the form:

$$u\,[\,v\,]_l \xrightarrow{c} u'\,[\,v'\,]_l \tag{1}$$

with u, v, u', v' some finite multisets of objects and l the label of a membrane. These rules are multiset rewriting rules that operate on both sides of the membranes, that is, a multiset u placed outside a membrane labeled by l and a multiset v placed inside the same membrane can be simultaneously replaced by a multiset u' and a multiset v' respectively. In this way, we are able to capture in a concise way the features of both communication rules (diffusion, binding, debinding etc ...) and transformation rules considered before. This generic type of rules was referred as *boundary rules* in [3].

We also associate to each rule a stochastic constant, c, which will be used to compute the probability of applying a rule in a given configuration, see [17]. This is necessary to characterize the *reality* of the phenomenon to be modeled. The necessity of taking into account these quantitative aspects has been made clear in a few recent studies regarding the use of P systems to model biological systems.

In what follows we will describe how to specify P systems models in the PRISM language.

First of all, since we work with continuous time P systems with Gillespie dynamics our model will be declared as a CTMC using the key word `stochastic`. The membranes occurring in the membrane structure will be represented using modules and the topology according to which membranes communicate or interact will be coded in the commands of each module.

Each module will describe the behavior of one membrane by representing the rules associated to it using commands and the objects placed in it using local variables.

More specifically, given $\Pi = (\Sigma, L, \mu, M_1, M_2, \ldots, M_n, R_1, \ldots, R_n)$ a P system as before, each membrane in μ will be uniquely identified with an identifier i, $1 \leq i \leq n$.

- Each membrane i will be specified using a module which will be called `compartment_i`.
- For each object $o \in \Sigma$ that can be present inside the compartment defined by membrane i a local variable `o_i` will be declared in module `compartment_i`. The initial value of the variable will be given by the corresponding initial multiset w_i; its value range will be determined experimentally or it will be derived from the literature.

  A constant `o_i_bound` representing the upper bound of the object `o_i` will be declared to specify the value range.
- To describe the rules in R_i commands will be used. We will focus on the generic type of rule in (1). In general, these rules need two membranes to interact in a synchronized way to exchange objects. In this case the two modules representing the corresponding membranes will synchronize the application of two different commands by using the label `rule_k`, where k is the index of the rule being specified.

  Therefore, assuming that compartment i is contained in compartment j and given a rule of the form

$$a_1, \ldots, a_p \, [\, b_1, \ldots, b_r \,]_i \xrightarrow{c} c_1, \ldots, c_s [\, d_1, \ldots, d_t \,]_i$$

  The command in module `compartment_j` will be:

```
[rule_k] a1_j > 0 & ... & ap_j > 0 &
         c1_j < c1_j_bound & ... & cs_j < cs_j_bound ->
         c * a1_j * ... *ap_j :
          (a1_j' = a1_j - 1) & ... & (ap_j' = ap_j - 1) &
          (c1_j' = c1_j + 1) & ... & (cs_j' = cs_j + 1);
```

  The command in module `compartment_i` will be:

```
[rule_k] b1_i > 0 & ... & br_i > 0 &
         d1_i < d1_i_bound & ... & dt_i < dt_i_bound ->
         b1_i * ... * br_i :
          (b1_i' = b1_i - 1) & ... & (br_i' = br_i - 1) &
          (d1_i' = d1_i + 1) & ... & (dt_i' = dt_i + 1);
```

Observe that these two commands are applied when the guards hold, that is, if and only if there are some reactants in the corresponding membranes and the products have not reached the upper bounds determined experimentally. Also note that the rate of this transition is the product of the individual rates:

$$(\texttt{c * a1\_j * ... * ap\_j}) * (\ \texttt{b1\_i * ... * br\_i})$$

Here we assume that the objects a's and b's are different, if we have an object with multiplicity greater than one present on the left hand side of the rule the rate associated to the command will be different and it will be computed as it is explained in [10].

When this transition is performed the local variables representing the reactants are decreased by one and the variables representing the products are increased by one.

Finally, note that although the rules of the general form in (1) require synchronization between two modules representing membranes, in the particular case of transformation, complex formation and dissociation rules only one membrane is involved and no synchronization is needed. Given a rule of type:

$$[\ a_1, \ldots, a_p\ ]_l \xrightarrow{c} [\ b_1, \ldots, b_r\ ]_l \tag{2}$$

the PRISM specification will be as follows:

```
[] a1_i > 0 & ... & ap_i > 0 &
   b1_i < b1_i_bound & ... & br_i < br_i_bound ->
      c * a1_i * ... * ap_i :
        (a1_i' = a1_i - 1) & ... & (ap_i' = ap_i - 1) &
        (b1_i' = b1_i + 1) & ... & (br_i' = br_i + 1);
```

4 Cell Cycle in Eukaryotes – A Case Study

In this section two different models of the cell cycle in eukaryotes will be presented and contrasted using PRISM and our simulator of P systems with Gillespie dynamics available from [25]. The first model is expressed using a P system specification with a single compartment whereas the second one [13] uses a π-calculus approach.

The cell division cycle in eukaryotes is a coordinated set of processes whereby a cell replicates all its components and divides into two nearly identical daughter cells. These processes are controlled by a complex network consisting of cyclin-dependent kinase (CDK) and its corresponding cyclin. Kinases, cdc14, and phosphatases, cdh1, regulate CDK activity. There are also stoichiometric inhibitors (CKI), that sequester cyclin-CDK dimers inhibiting their activity.

4.1 A P System Model

Our P system model is given by,

$$\Pi = (\Sigma, \{1\}, [\ ], (1, w), R)$$

where:

- Σ contains all the protein and complexes of proteins involved in the system: $\Sigma = \{cdk,\ cyclin,\ cdk.cyclin,\ cdk.degc,\ cki, cdk.cyclin.cki,\ cdh1,\ cdh1off,\ cdc14,\ cdc14off\}$
- the membrane structure has only one component that will be labeled by 1.
- w is the initial multiset of objects (molecules) which determines the initial configuration of the system,

$$w = cdk^{100}\ cyclin^{200}\ cki^{100}\ cdh1^{100}\ cdc14^{200}$$

- the rules of R describe the interactions taking place in the cell cycle control[2]:
  $r_1 : [\ cdk,\ cyclin\ ]_1 \xrightarrow{c_1} [\ cdk.cyclin\ ]_1 \quad c_1 = 0.5$
  cdk is activated upon binding with its corresponding $cyclin$ producing the dimer $cdk.cyclin$.
  $r_2 : [\ cdh1,\ cdk.cyclin\ ]_1 \xrightarrow{c_2} [\ cdh1,\ cdk.degc\ ]_1 \quad c_2 = 0.005$
  $r_3 : [\ cdk.degc\ ]_1 \xrightarrow{c_3} [\ cdk\ ]_1 \quad c_3 = 0.001$
  These two rules describe how $cdh1$ regulates the activity of the dimers $cdk.cyclin$ by degrading the bound cyclin.
  $r_4 : [\ cdk.cyclin,\ cki\ ]_1 \xrightarrow{c_4} [\ cdk.cyclin.cki\ ]_1 \quad c_4 = 0.003$
  $r_5 : [\ cdk.cyclin.cki\ ]_1 \xrightarrow{c_5} [\ cdk.cyclin,\ cki\ ]_1 \quad c_5 = 0.3$
  cki also inhibits the activity of the dimers $cdk.cyclin$ by forming the triplets $cdk.cyclin.cki$.
  $r_6 : [\ cdh1,\ cdk.cyclin\ ]_1 \xrightarrow{c_6} [\ cdh1off,\ cdk.cyclin\ ]_1 \quad c_6 = 0.005$
  $cdh1$ can also be inhibited by $cdk.cyclin$ dimers.
  $r_7 : [\ cdh1off,\ cdc14\ ]_1 \xrightarrow{c_7} [\ cdh1,\ cdc14off\ ]_1 \quad c_7 = 0.009$
  $r_8 : [\ cdh1,\ cdc14off\ ]_1 \xrightarrow{c_8} [\ cdh1,\ cdc14\ ]_1 \quad c_8 = 0.009$
  $cdh1$ and $cdc14$ regulate each other activity by inhibition and activation.

Now we will specify the previous P system in the PRISM language using the algorithm described in section 3. Since Π is a continuous time P system with Gillespie dynamics, our model will be declared as being `stochastic`.

The PRISM model will have a single module, `compartment`, representing the only membrane present in the membrane structure of Π.

```
module compartment
     ⋮
endmodule
```

In this module we will describe the alphabet Σ and the initial multiset w using local variables. For example, cdk, $cyclin$ and the dimer $cdk.cyclin$ are specified as follows:

```
cdk_1 : [ 0 .. cdk_1_bound ] init 100;
cyclin_1 : [ 0 .. cyclin_1_bound ] init 200;
cdk.cyclin_1 : [ 0 .. cdk.cyclin_1_bound] init 0;
```

[2] The time units of the stochastic constants associated with the rules are minutes.

Finally the rules from R will be described using commands. We observe that since the model consists only of rules of the form in (2) no synchronization is required in this case study.

$[cdk,\ cyclin]_1 \xrightarrow{c_1} [cdk.cyclin]_1 \quad c_1 = 0.5$

```
[] cdk_1 > 0 & cyclin_1 > 0  &
   cdk.cyclin_1 < cdk.cyclin_1_bound ->
       c1*cdk_1*cyclin_1 :
         (cdk_1' = cdk_1 - 1) & (cyclin_1' =  cyclin_1 - 1) &
         (cdk.cyclin_1' = cdk.cyclin_1 + 1);
```

$[cdk.cyclin.cki]_1 \xrightarrow{c_5} [cdk.cyclin,\ cki]_1 \quad c_5 = 0.3$

```
[] cdk.cyclin.cki_1 > 0 &
   cdk.cyclin_1 < cdk.cyclin_1_bound & cki_1 < cki_1_ bound ->
       c5*cdk.cyclin.cki_1 :
         (cdk.cyclin.cki_1' = cdk.cyclin.cki_1  - 1) &
         (cki_1' = cki_1 + 1) &
         (cdk.cyclin_1' = cdk.cyclin_1 + 1);
```

We have run simulations of the previous P system and PRISM specifications using our simulator of P system with Gillespie dynamics and the PRISM simulator. In Figure 1 we depict the evolution of the number of objects representing the dimer *cdk.cyclin*, *cdh*1 and *cdc*14.

Observe that, although both simulators use the same strategy for the evolution of the two different models, the simulations are not exactly the same. This is due to the fact that we are dealing with stochastic approaches.

Nonetheless, both runs show a sudden increase of the dimer *cdk.cyclin* reaching a peak of almost 100 molecules in less than a minute, then the number of dimers decay steadily to zero. The evolution of *cdh*1 and *cdc*14 is similar, at the beginning both decay slightly and then they increase reaching a number of molecules approximately equal to the initial one.

4.2 A π-Calculus Model

Within the framework of π-calculus a system of interacting molecular entities is described and modeled by a system of concurrent communicating processes. Communication occurs on complementary channels, that are identified by specific names. Each molecule in the molecular system is described by a process and modifications due to chemical interactions are represented using communication and message passing between different processes through different channels, [20].

In what follows we comment on some fragments of a π-calculus specification of the same network of the cell cycle specified before using P systems. These fragments were taken from [13].

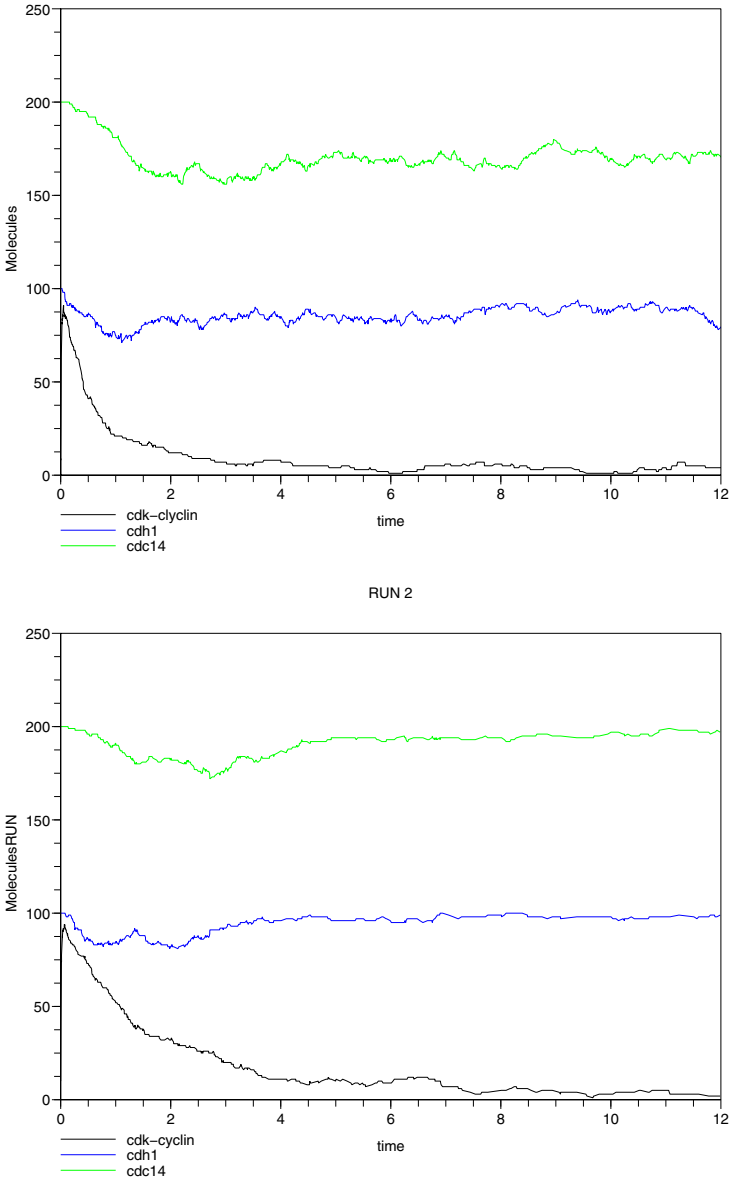


Fig. 1. Simulations using the PRISM simulator (first graph) and our simulator of P systems with Gillespie dynamics (second graph)

1. `SYSTEM ::= CYCLIN | CDK | CDH1 | CDC14 | CKI`

   First the molecular population is specified as a system of concurrent processes each one representing a molecule; `CYCLIN`, `CDK`, `CDH1`, `CDC14` and `CKI`.

The following two fragments are examples of how chemical interactions are specified in π-calculus.

2. `CYCLIN ::= (v bb) BINDING-SITE`
   `BINDING-SITE ::=` $(\overline{lb}\langle$`bb`$\rangle, \mathtt{R}_4)$`, CYCLIN-BOUND`
   `CDK ::= (lb(cbb),` $\mathtt{R}_4$`). CDK-CATALYTIC`

   The process `CYCLIN` is defined as another process `BINDING-SITE` with communication channel `bb`. The process `CDK` has the complementary channel to `BINDING-SITE`, $\overline{lb}\langle$`bb`$\rangle$, and so they can communicate, with rate $\mathtt{R}_4$, to produce two new processes `CYCLIN-BOUND` and `CDK-CATALYTIC`. This fragment correspond to rule r_1 of the P system specification.

3. `CKI ::= DEGRCKI + BINDCYC`
   `BINDCYC ::= (bind(x),`$\mathtt{R}_{11}$`).0`
   `CYC-CDK-CKI ::=` $(\overline{bind}\langle$`bb`$\rangle, \mathtt{R}_{11})$`.TRIM`

   This fragment is similar to the previous one. `CKI` can be replaced either by the process `DEGRCKI` or `BINDCYC` nondeterministically. The process `BINDCYC` can communicate with `CYC-CDK-CKI` to produce `TRIM`, process representing the trimer. This fragment correspond to rule r_4 of the P system specification.

The above π-calculus specification may be translated into PRISM [22]. In the next page the parts corresponding to the fragments of the specification presented before are given. The π-calculus specification was mapped into PRISM using modules to describe processes, labels of commands correspond to communication channels, values of variables represent the number of different processes and commands specify the effect of a communication.

```
module cyclin
    cyclin : [0..CYCLIN] init CYCLIN;
    cyclin_bound : [0..CYCLIN] init 0;
    trim : [0..CYCLIN] init 0;
    [lb] cyclin>0 & cyclin_bound<CYCLIN
        -> cyclin : (cyclin_bound'=cyclin_bound+1) &
                    (cyclin'=cyclin-1);
    [bind] cyclin_bound>0 & trim<CYCLIN
        -> cyclin_bound : (trim'=trim+1) &
                          (cyclin_bound'=cyclin_bound-1);
endmodule
```

The module above describes the processes representing molecules of type cyclin. The variables `cyclin`, `cyclin_bound` and `trim` represent the number of processes `CYCLIN`, `CYCLIN_BOUND` and `TRIM` of the previous π-calculus specification. Below the modules describing the molecules of type `cdk` and `cki` are presented.

Observe that the command labels `lb` and `bind` specify the communication channels and the effect of these communications are described in the

corresponding commands. For example, the label `lb` synchronise the first commands in modules `cyclin` and `cdk`, where the number of processes of type `cyclin` is decreased by one and the number of processes of type `cyclin_bound` and `cdk_cat` are increased by one representing the removal of a process `CYCLIN` and the creation of two new processes `CYCLIN_BOUND` and `CDK-CATALYTIC`.

```
module cdk
    cdk : [0..CDK] init CDK;
    cdk_cat : [0..CDK] init 0;
    [lb] cdk>0 & cdk_cat<CDK
        -> cdk : (cdk_cat'=cdk_cat+1) & (cdk'=cdk-1);
endmodule
```

The fragment 3 of the π-calculus specification is described in the next module and the creation of a process `TRIM` is represented using the commands labelled by `bind` which decrease by one the variable `cki` representing the number of `CKI` processes whereas in the module `cyclin` the variable `trim` is increased by one.

```
module cki
    cki : [0..CKI] init CKI;
    [bind] cki>0 -> cki : (cki'=cki-1);
endmodule
```

Using the PRISM simulator simulations of the previous π-calculus model can be obtained [22]. The evolution of the number of processes `CYCLIN`, `CDH1` and `CDC14` is depicted below. Note the similarity between this graph and the ones in Figure 1.

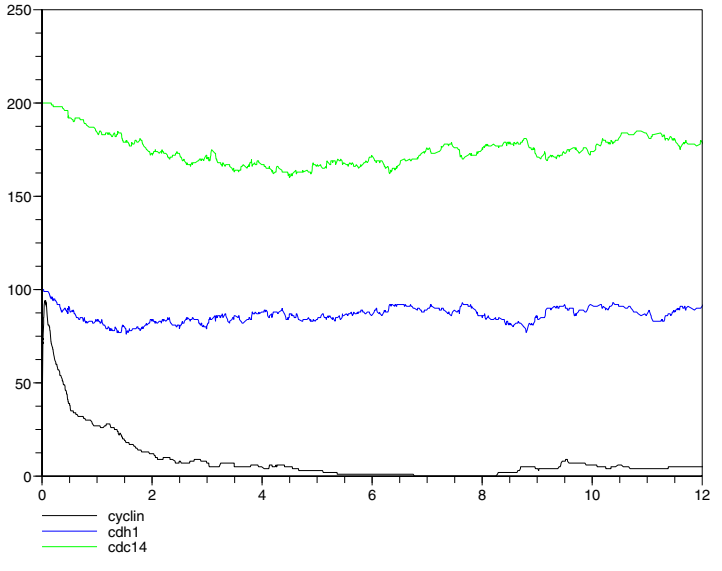


Fig. 2. A Simulation of the π-calculus model

5 Some Results and Discussions

We deal with systems of interacting biochemical entities. In this section we will show how these systems are modeled using π-calculus and P systems formalism and how these are represented in PRISM.

From the previous case study modeled by using π-calculus and P systems approaches we can identify some general principles of representing, in the formalisms, different aspects of the modeling process.

In P systems, molecules are represented using objects from a finite alphabet and therefore the molecular population present in the system is specified by multisets of objects. In π-calculus a concurrent and communicating process abstracts the behavior of a molecule and the whole molecular system is specified as a system of concurrent and communicating processes.

Compartments are explicitly specified in P systems as regions delimited by membranes. Although in π-calculus there is no component that corresponds directly with a biological compartment, processes representing molecules in the same compartment share certain exclusive communication capabilities (private communication channels), that are inaccessible to processes representing molecules in other compartments.

Regarding biochemical reactions, in P systems they are specified using rewriting rules according to which objects representing reactants are replaced with objects representing products. In the π-calculus approach different processes representing molecules interact through complementary communication channels; this communication and the changes triggered by it describe a chemical interaction between the molecules represented using these processes.

The translocation of a molecule from one compartment to another one is modeled in P system using a particular type of rewriting rule, called boundary rule, where the compartments involved in the movement are specified. In the case of π-calculus mobile communication in which communication channel names are sent as messages is used to describe the movement of a process from one compartment to another using the extrusion of a private channel's scope.

In the following two tables we sum up how biomolecular systems are specified in P systems and in π-calculus and how these specifications can be mapped into PRISM so that we can perform probabilistic model checking on them.

Biomolecular entity	**P system entity**	**π-calculus entity**
Molecule	Object	Process
Molecular Population	Multiset of objects	System of concurrent processes
Compartment	Region defined by a membrane	Private Communication channels
Biochemical Transformation	Re-writing rule	Communication through channels
Compartment Translocation	Boundary rule	Extrusion of a private channel's scope

Biomolecular entity	**P system PRISM**	**π-calculus PRISM**
Molecule	Variable	Module
Molecular Population	Variable values	Variables in modules
Compartment	Module	Command Labels
Biochemical Transformation	Command	Synchronization between commands
Compartment Translocation	Synchronization between commands	Synchronization between commands

One of the key features of our approach is the explicit stochastic behavior of our models. As mentioned before stochastic systems defy conventional intuition and consequently are harder to conceive. The main stream methodology to get the statistics of a stochastic system is the simulation of many trajectories. Nevertheless, simulation is the exploration of finite behaviors over given time intervals, whereas probabilistic model checking allows us to investigate the truth or otherwise of temporal queries expressed in temporal logics over possibly infinite sets of behaviors over possibly unbounded time intervals [6].

In what follows we use CSL (Continuous Stochastic Logic) and PRISM to formulate and check three temporal biological queries against our model of the cell cycle in eukaryotes in order to illustrate what kind of properties may be checked.

First, we study the expected number of molecules of *cdk.cyclin*, *cdh*1 and *cdc*14. For this we associate to each state a reward representing the value of the variable which specifies the corresponding protein. Then we formulate the query regarding the expected value of that reward at the time instant `T`:

```
R = ? [ I = T ]
```

In figure 3 we have plotted these expected values as T varies.

Observe that the expected number of molecules of dimers *cyclin.cdk* decreases steadily to low numbers whereas *cdh*1 and *cdc*14 decrease slightly during the first two minutes to start increasing gradually afterwards to reach approximately the initial number of molecules.

In order to get a finer grain analysis of the monotonic decrease of *cyclin.cdk* we investigate the probability that the dimers `cdk.cyclin` at time `T` is greater than a given threshold `t`. This property is specified by the CSL formula:

```
P = ? [ true U[T,T] cdk.cyclin >= t ]
```

In Figure 4 it is depicted the probability that the number of dimers `cdk.cyclin` is greater than 90, 80 and 70 as time varies for the P system specification.

Besides transient analysis we can also study steady state properties using the operator `S`. In order to verify that *cdh*1 and *cdc*14 return to their initial values in the long term we have verified the following temporal queries:

```
S >= 0.9 [ cdh1 = 100  ]
S >= 0.9 [ cdc14 = 200  ]
```

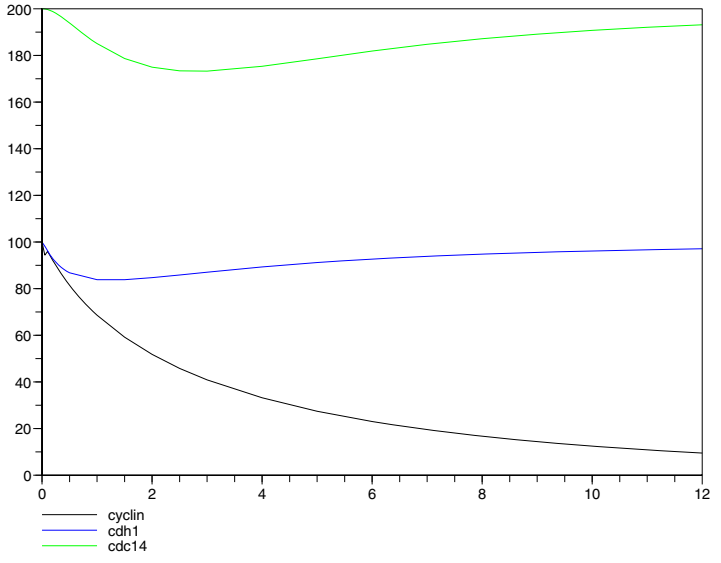


Fig. 3. Expected evolution of the P system specification

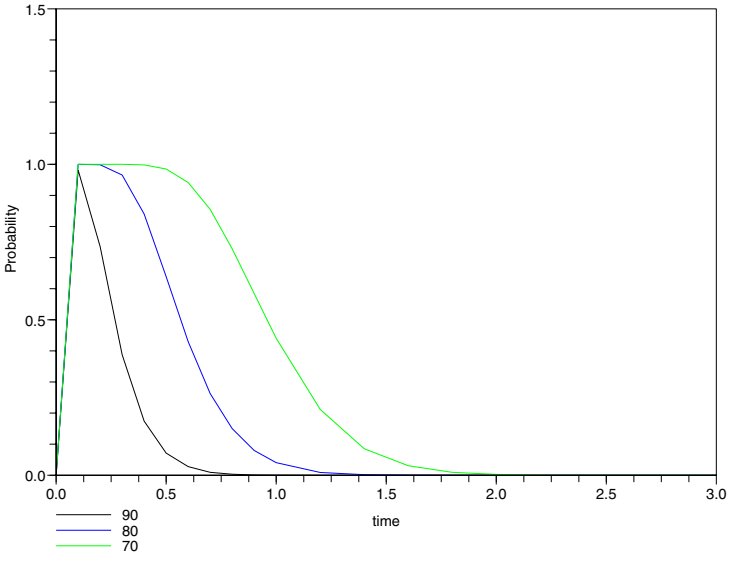


Fig. 4. Probability that the number of dimers cdk.cyclin is greater than a given threshold in the P system specification

6 Conclusions and Future Work

In this paper we have discussed how P systems and π-calculus can model systems of interacting biochemical entities. A simple case study regarding the cell cycle has been used to illustrate the different methodologies. We have also shown how P systems and π-calculus specifications can be translated into PRISM allowing us to perform probabilistic model checking; in this respect specific questions were addressed for the P system specification.

Future work will aim to compare through more complex case studies some differences between P system specifications and other modeling paradigms - Petri nets, cellular automata, ambient calculus, in order to understand advantages and limitations offered by these methods and their suitability for different problems.

Acknowledgement

This work is supported by Ministerio de Ciencia y Tecnología of Spain, by *Plan Nacional de I+D+I* (TIN2005-09345-C04-01), cofinanced by FEDER funds, by Junta de Andalucía, by project of Excellence TIC 581, and by a FPU fellowship from the Ministerio de Ciencia y Tecnología of Spain.

References

1. Alur, R., Henzinger, T.A. (1999) Reactive Modules, *Formal Methods in System Design*, **15**, 7–48.
2. Andrei, O., Ciobanu, G., Lucanu, D. (2005) Executable Specifications of P Systems, *LNCS*, **3365**, 126–145.
3. Bernardini, F., Manca, V. (2003) P Systems with Boundary Rules, *LNCS*, **2597**, 107–118.
4. Bernardini, F., Gheorghe, M. (2004) Population P Systems, *J. UCS*, **10**(5), 509–539.
5. Bianco, L., Fontana, F., Manca, V. (2006) P Systems with Reaction Maps, *International Journal of Foundations of Computer Science*, **17** (1), 27–48.
6. Calder, M., Vyshemirsky, V., Gilbert, D, Orton, R. Analysis of Signalling Pathways using Continuous Time Markov Chains, *Transactions on Computational Systems Biology*, to appear.
7. Dang, Z., Ibarra, O.H. (2005) On One-Membrane P systems Operating in Sequential Mode, *Int. J. Found. Comput. Sci.*, **16** (5), 867–881.
8. Fontana, F., Bianco, L., Manca, V. (2005) P Systems and the Modeling of Biochemical Oscillations, *LNCS*, **3850**, 199 – 208.
9. Gillespie, D.T. (1976) A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions, *J Comput Physics*, **22**, 403–434.
10. Gillespie, D.T. (1977) Exact Stochastic Simulation of Coupled Chemical Reactions, *The Journal of Physical Chemistry*, **81** (25), 2340–2361.
11. Goss, P.J.E., Peccoud, J. (1998) Quantitative modeling of stochastic systems in molecular biology using stochastic Petri nets., *Proc. Natl. Acad. Sci. USA*, **95**, 6750–6755.

12. Krasnogor, N., Gheorghe, M., Terrazas, G., Diggle, S., Williams, P., Camara, M. (2005) An appealling Computational Mechanism Drawn from Bacterial Quorum Sensing, *Bulletin of the EATCS*, **85**, 135–148.
13. Lecca, P., Corrado, P. Cell Cycle Control in Eukaryotes: A BioSpi Model, *Electronic Notes in Theoretical Computer Science*, to appear.
14. Novak, B., Csikasz-Nagy, A., Gyorffy, B., Nasmyth, K., Tyson, J.J. (1998) Model Scenarios for Evolution of the Eukaryotic Cell Cycle, *Phil. Trans. R. Soc. Lond.*, **353**, 2063–2076.
15. Păun, Gh. (2000) Computing with Membranes, *Journal of Computer and System Sciences*, **61**(1) 108–143.
16. Pérez-Jiménez, M.J., Romero-Campero, F.J. Modelling Gene Expression Control Using P Systems: The Lac Operon, A Case Study, submitted.
17. Pérez-Jiménez, M.J., Romero-Campero, F.J. (2006) P Systems, a New Computationl Modelling Tool for Systems Biology, *Transactions on Computational Systems Biology VI, LNBI* **4220**, 176–197.
18. Pérez-Jiménez, M.J., Romero-Campero, F.J. A Model of the Quorum Sensing System in Vibrio Fischeri Using P Systems, submitted.
19. Pescini, D., Besozzi, D., Mauri, G., Zandron, C. (2006) Dynamical Probabilistic P Systems, *International Journal of Foundations of Computer Science*, **17** (1), 183–195.
20. Regev, A., Shapiro, E. (2004) The π-Calculus as an Abstraction for Biomolecular Systems, In G. Ciobanu and G. Rozenberg, editors, *Modelling in Molecular Biology.* Springer, Berlin.
21. Terrazas, G., Krasnogor, N., Gheorghe, M., Bernardini, F., Diggle, S., Camara, M. (2005) An Environment Aware P-System Model of Quorum Sensing. *CIE 2005, LNCS*, **3526**, 473–485.
22. PRISM Web Site: `http://www.cs.bham.ac.uk/~dxp/prism/`
23. The P Systems Web Site: `http://psystems.disco.unimib.it`
24. SciLab Web Site `http://scilabsoft.inria.fr/`
25. P System Simulator: `http://www.dcs.shef.ac.uk/~marian/PSimulatorWeb/PSystemMF.htm`

Graphical Modeling of Higher Plants Using P Systems

Alvaro Romero-Jiménez, Miguel A. Gutiérrez-Naranjo, and Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, Spain
Alvaro.Romero@cs.us.es, magutier@us.es, marper@us.es

Abstract. L systems have been widely used to model and graphically represent the growth of higher plants [20]. In this paper we continue developing the framework introduced in [21], which make use of the topology of membrane structures to model the morphology of branching structures.

1 Introduction

The growth of plants, considered as a function of time, have attracted the attention of scientific community for a long time. Features such as the bilateral symmetry of leaves, the central symmetry of flowers and, more recently, the study of self-similarity and fractal structure have been matter of study for computer scientists, mathematicians, and life scientists among others.

In 1968, Aristid Lindenmayer presented a theoretical framework for studying the development of simple multicellular organisms. The devices introduced in this framework are known as *parallel rewriting systems* or *L systems*.

L systems were introduced for modeling multicellular organisms in terms of division, growth, and death of individual cells [10,11]. These organisms are treated as an assembly of discrete units, which represent the individual cells. These systems must be considered as dynamic models, which means that the form of the organism is the result of development along time. This development is described in terms of *production rules*, which are applied in parallel and are intended to capture the simultaneous progress of time in all parts of the growing organisms.

Several years later, the range of applications of L systems were extended to higher plants and complex branching structures [3,4]. In the first approach, the essence of development of less complex organisms is the replacement of individual cells by sets of cells, according to the production rules of the system. On the other hand, the units of information in L systems modeling higher plants represent complex structures, such as branches or leaves, instead of individual cells. These structures are replaced by other ones using the production rules.

In [5,6] a first approach for using P systems to simulate the growth and development of living plants is presented. This approach mixes L systems and

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 496–506, 2006.

P systems, being in fact an L system "factorized" into several units, which are then computed in the compartments delimited by the membranes of the P system.

L systems use strings as data structures, which fits in a natural way in sequential structures such as microorganisms, or linear structure of fractals such as the Koch curve [8,9]. Nonetheless, the visual interpretation of strings of symbols as branching structures needs to add memory pointers in order to remember the location and orientation in which the branches were developed. These memory facilities are the key for developing several branches from the same point.

The topology of P systems is inherently a branching structure based on the inclusion relationship. This feature allowed us to present a framework for modeling the topology of living plants, without the necessity of considering memory pointers [21]. We think our approach is closer to reality than L systems, in the sense that we do not make "rewriting" over the membrane structure, but instead we use evolution rules to expand it. This is inspired by the fact that mature structures in higher plants, such as trunks and branches, keep their morphology along time. They change only in length and width, and the growth of new structures (leaves, flowers, new branches, and so on) is started only from specific points, already present.

In this paper we continue developing the framework introduced in [21], providing two means (randomness and non-determinism) for modeling the individual variations among different specimens of a same species that occur in nature.

The paper is organized as follows: first L systems and the usual way to visualize them are recalled in Sections 2 and 3. In Section 4, the variant of P systems used in this paper, a restricted version of P systems with membrane creation, is presented. Section 5 is devoted to the graphical visualization of the configurations of these P systems, whereas we discuss in Section 6 the differences between the representations obtained when stochastic and non-deterministic P systems are considered. Finally, conclusions and lines for future research are presented.

2 L Systems

The key idea of L systems for formalizing the development of plants is that of rewriting. This is a technique for defining complex objects by successively replacing parts of a simple initial object by using *production* rules.

The first formal definition of rewriting systems operating on strings of symbols was proposed by Thue at the beginning of the twentieth century (see [22]), but rewriting systems started to be widely considered after Chomsky's work on formal grammars [2], where the concept of rewriting is used to describe natural languages.

The essential difference of L systems with respect to Chomsky grammars lies in the method of applying the production rules. In Chomsky grammars, production rules are applied sequentially, whereas in L systems they are applied in parallel: in a derivation step all symbols of the string are rewritten.

The simplest class of L systems are the deterministic and context-free ones. Let V denote an alphabet, V^* the set of all words over V, and V^+ the set of all nonempty words over V. A *string 0L system* is an ordered triplet $G = \langle V, \omega, P \rangle$ where V is the *alphabet* of the system, $\omega \in V^+$ is a nonempty word called the *axiom* and $P \subset V \times V^*$ is a finite set of *production rules*. A production rule (a, v) is written as $a \to v$. The letter a and the word v are called the predecessor and the successor of this production rule, respectively. It is assumed that for any letter $a \in V$, there is at least one word $v \in V^*$ such that $a \to v \in P$. A 0L system is *deterministic* (noted D0L system) if and only if for each $a \in V$ there is exactly one $v \in V^*$ such that $a \to v \in P$.

Let $\mu = a_1 \dots a_m$ be an arbitrary word over V. The word $\rho = \phi_1 \dots \phi_m$, with $\phi_1 \dots \phi_m \in V^*$, is directly derived from (or generated by) μ, and we denote it by $\mu \Rightarrow \rho$, if and only if $a_i \to \phi_i \in P$ for all $i \in \{1, \dots, m\}$. Starting with the axiom $\omega \in V^+$, a sequence of strings $\mu_0 = \omega, \mu_1, \mu_2, \dots$ is generated recursively, where $\mu_i \Rightarrow \mu_{i+1}$, that is, the string μ_{i+1} is obtained from the preceding string μ_i by replacing *simultaneously* every symbol in μ_i according to production rules of the system.

3 Graphical Representation of L Systems

Originally, L systems were conceived for the study of multicellular organisms and the neighborhood relations between their different cells. After the incorporation of geometric features, L systems became appropriate for computer graphics representation that allows visualizations of these multicellular organism and their developmental processes. The first steps in this line can be found in the eighties' literature [17,23], but the most popular graphical interface for L systems was introduced by P. Prusinkiewicz [18,19] based on the previous Papert's concept of *turtle graphics* [14]. In an informal description, we can consider a turtle standing on a sheet of paper facing a given direction. The tail of the turtle is full of ink and it traces a line on the sheet when the turtle moves. The turtle obeys several commands: move forward by a fixed length l drawing or not the corresponding segment; turn left or right by a fixed angle δ.

More formally, a *state* of the turtle is defined as a triplet (x, y, α), where (x, y) represent the Cartesian coordinates of the turtle's position and the angle α represent the direction in which the turtle is facing. Given a step size l and an angle increment δ, the turtle responds to the following commands:

- F: the state of the turtle changes from (x, y, α) to $(x + l\,cos\alpha, y + l\,sen\alpha, \alpha)$. A line segment between (x, y) and $(x + l\,cos\alpha, y + l\,sen\alpha)$ is drawn.
- f: analogous to the previous command, the state of the turtle changes from (x, y, α) to $(x + l\,cos\alpha, y + l\,sen\alpha, \alpha)$. The segment is not drawn.
- $+$: the state of the turtle changes from (x, y, α) to $(x, y, \alpha + \delta)$.
- $-$: the state of the turtle changes from (x, y, α) to $(x, y, \alpha - \delta)$.

This method has been profusely used to interpret strings. For example, the representation of the string $F + F - -F + F$ with initial state (0, 0, 0), $l = 2$ cm and $\delta = 60$ degrees is depicted in Figure 1.

Fig. 1. The string $F + F - -F + F$

According to these rules, the turtle interprets a character string as a sequence of line segments. Note that different strings can lead to the same graphical representation.

If we want to model tree-like shapes and branching structures then new features have to be added. For that, an extension of turtle interpretation to strings with brackets is considered. Two new symbols are introduced to delimit a branch: the symbols "[" and "]". The *turtle interpretation* of the symbols is the following, when [is read, the turtle should remember its current direction and position. Then the branch can be drawn by the usual interpretation. Termination of the branch is marked by]. The turtle must then return to the location of the branch point, which it remembers. The formal interpretation of the symbols is the following.

- [: push the current state of the turtle onto a push-down stack. The information saved on the stack contains the turtle's position and orientation, and possibly other attributes such as the color and width of lines being drawn.
-] : pop a state from the stack and make it the current state of the turtle. No line is drawn, although in general the position of the turtle changes.

For example, the representation of the string $F[+F[+F[+F][F]][F[F][-F]]]$ $[F[F[+F][F]][-F[+F][F]]]$ with initial state (0, 0, 90), $l = 2$ cm and $\delta = 22.5$ degrees is shown in Figure 2.

4 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[1].

[1] A detailed description of P systems can be found in [15] and updated information in [24].

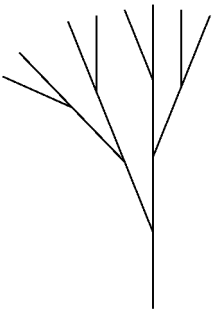


Fig. 2. The string $F[+F[+F[+F][F]][F[F][-F]]][F[F[+F][F]][-F[+F][F]]]$

In this paper we will consider P systems which make use of membrane creation rules, which was first introduced in [7,12]. 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 $\Pi = (O, \mu, w_1, \ldots, w_m, R)$ where:

1. O is the alphabet of *objects*.
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. $w_1, \ldots, w_m$ 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 (extending the membrane structure μ) together with the objects contained in the regions defined by its membranes constitute a configuration of the system. A computation 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 maximal parallel way; a rule in a region is applied if and only if the object occurring on 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.

5 Graphical Representation of Restricted P Systems with Membrane Creation

In this section we show how to use, through a suitable graphical representation, restricted P systems with membrane creation to model branching structures.[2] 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. It seems therefore a perfect frame to encode the branching structure.

Once we have a membrane structure establishing the topology of the object we want to model, we will follow a variant of the turtle interpretation of L systems. Let us suppose that the alphabet O of objects contains the objects $F, +$ and $-$ and let us fix the length l and the angle δ.

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 branch is lengthening. This segment is drawn rotated with respect to the segment corresponding to the parent membrane with an angle of $n \times \delta$, 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.

In the real world, the branches of a plant not only lengthen but also widen themselves. Thus, we also fix the width w and use a new symbol W whose multiplicity will specify 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 $n \times w$.

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

- The alphabet of objects is $O = \{L, E, W, F, +, -, B_L, B_R, B_{S_1}, B_{S_2}\}$.
- The initial membrane structure together with the initial multiset of objects is $[LEWFB_LB_{S_1}]$.
- The rules are:

$$B_{S_1} \to [LEWFB_{S_2}B_R] \qquad B_L \to [+LEWFB_LB_{S_1}]$$
$$B_{S_2} \to [LEWFB_LB_{S_1}] \qquad B_R \to [-LEWFB_LB_{S_1}]$$
$$L \to LF \qquad E \to EW$$

In this system, the objects B_{S_1} and B_{S_2} represent 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 and E do not have a graphical interpretation; they can be considered as seeds for growing the branch in length and width.

[2] A preliminary version with simpler examples can be found in [21].

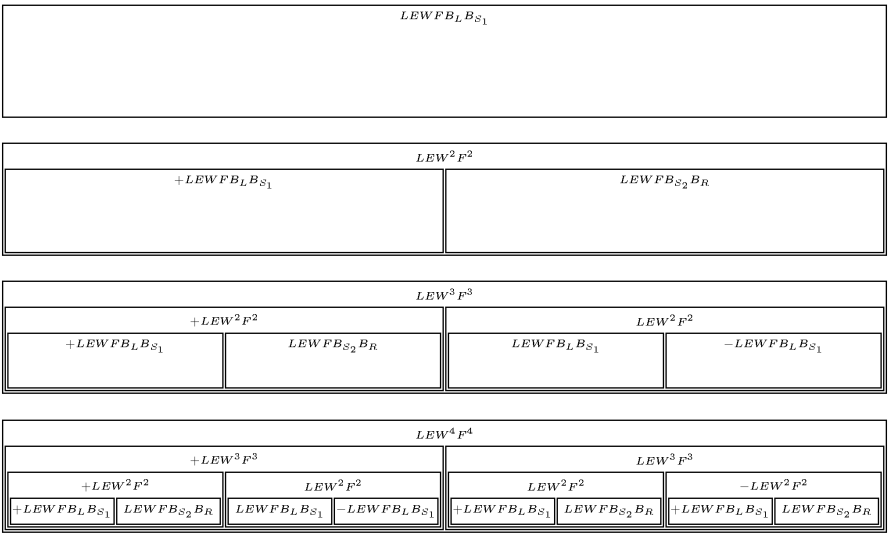


Fig. 3. First four configurations

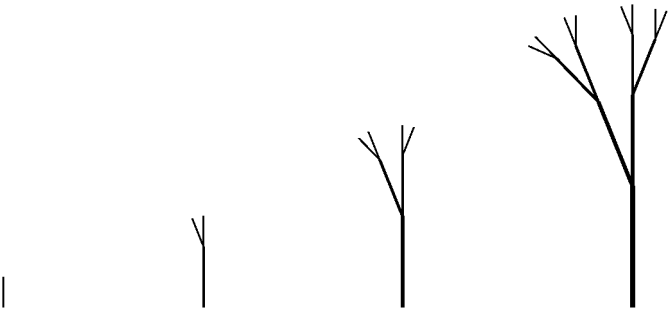


Fig. 4. Graphical representation of the configurations

This way, a thorough analysis of the initial membrane structure and of the rules shows that Π_1 models a branching structure consisting of a main trunk from which branches to the left and to the right come out alternatively. These new branches behave in the same way as the main trunk. Figure 3 shows the evolution of the system along three computation steps, and we can see in Figure 4 the corresponding graphical representation of each configuration, where we fix a bottom-up orientation with a length l of 1 cm, a width w of 0.4 pt and an angle δ of 22.5 degrees.

Note how the graphical representation of the configurations shows that the growth of the branching structure being modeled is not made by replacing segments by complex and repetitive modules, but by expanding the figure from

specific points with new segments (similar to how branches of higher plants spring from buds).

6 Stochastic Versus Non-deterministic P Systems

In the previous section we have introduced a technique to construct somehow realistic representations of branching structures in the framework of P systems. Each configuration of a given restricted P system with membrane creation can be translated to a set of graphical commands which produce the required picture.

An important drawback, as the example above shows, is that all the trees generated by the same deterministic P system are identical. It is then necessary to introduce specimen-to-specimen variations that preserve the general aspects of the branching structure but modify its details.

One possible way to achieve this is to consider stochastic P systems. Several alternatives to incorporate randomness into membrane systems can be found in the literature (see [1,13,16] and the references therein). One of them is to associate each rule of the P system with a probability. 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.

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

- The alphabet of objects is $O = \{L, E, W, F, +, -, T, B\}$.
- The initial membrane structure together with the initial multiset of objects is $[LEWFTB]$.
- The rules are:

$$T \to [LEWFTB] \qquad L \to LF$$
$$B \xrightarrow{2/3} [+LEWFTB] \qquad E \to EW$$
$$B \xrightarrow{1/3} [-LEWFTB]$$

Note that we do not make explicit the probability of the rule when this is one. Also, it is easy to see from the probabilities assigned to the rules with left–hand side equal to B, that the trees generated by this system favors developing branches to the left.

Let us fix a length l of 1 cm, a width w of 0.4 pt and an angle δ of 22.5 degrees. Thus, after two computation steps we could have obtain, depending on the rules chosen for the object B, any of the trees shown in Fig. 5, with the upper left one being the most probable, and this probability decreasing as we go from left to right and from top to bottom.

On the other hand, for non-deterministic P systems we could consider together all the trees generated by all of its computations. We would obtain in this way a "forest". Thus, taking Π_2 as a non-deterministic system instead of a stochastic one, we would obtain after two computation steps the graphical representation

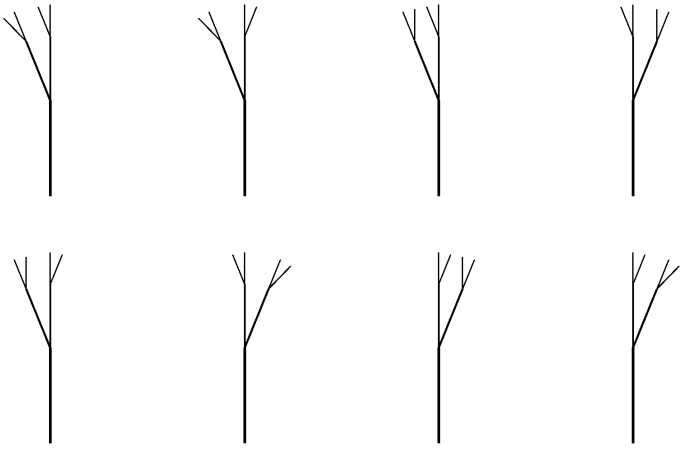


Fig. 5. Trees obtained from Π_2

depicted in Figure 5 (here the trees have been arranged in a 2×4 rectangular grid, but another methods to place the trees are possible).

7 Final Remarks

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.

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.

It is also fundamental to develop computer tools that make easier the generation, within our framework, of the graphical representations from given models. For that, it should be interesting to analyze the possibility of transforming the models of higher plants constructed using membrane computing into equivalent ones using Lindenmayer systems. This way, we could reuse the available software for these latter systems.

The computation model considered here, restricted P system with membrane computing, is a very simple one (at least, in terms of membrane computing). We finish the paper by proposing extensions to this model in several ways that we think are interesting enough to be considered and studied:

- A labeling of the membranes could be useful to distinguish between different parts of the plant being modeled.
- The use of communication rules, allowing objects to cross the membranes of the system, are basic for modeling the flow of nutrients and hormones.
- Rules of the form $o \rightarrow \mu$, where o is an object and μ is a membrane structure, could lead to a clearer, faster and more compact representation of a plant.

Acknowledgement. This work was supported by the Project TIN2005-09345-C03-01 of the Ministry of Education and Science of Spain, cofinanced by FEDER funds, and by the Project of Excellence TIC-581 of the Junta de Andalucía.

References

1. Ardelean, I.I., Cavaliere, M.: Modelling Biological Processes by Using a Probabilistic P System Software. *Natural Computing*, **2**, 2 (2003), 173–197.
2. Chomsky, N.: Three Models for the Description of Language. *IRE Trans. on Information Theory*, **2**, 3 (1956), 113–124.
3. Frijters, D., Lindenmayer, A.: A Model for the Growth and Flowering of Aster Novae-Angliae on the Basis of Table (0,1)L systems. In Rozenberg, G., Salomaa, A. (eds.): *L systems. Lecture Notes in Computer Science*, Vol. 15. Springer–Verlag, Berlin, 1974, 24–52.
4. Frijters, D., Lindenmayer, A.: Developmental Descriptions of Branching Patterns with Paracladial Relationships. In Lindenmayer, A., Rozenberg, G. (eds.): *Automata, Languages, Development*. North–Holland, 1976, 57–73.
5. Georgiou, A., Gheorghe, M.: Generative Devices Used in Graphics. In Alhazov, A., Martín–Vide, C., Păun, Gh. (eds.): *Preproceedings of the Workshop on Membrane Computing*. Technical Report, Vol. 28/03. Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona, 2003, 266–272.
6. Georgiou, A., Gheorghe, M., Bernardini, F.: Membrane–Based Devices Used in Computer Graphics. In Ciobanu, G. Păun, Gh., Pérez–Jiménez, M.J. (eds.): *Applications of Membrane Computing*. Springer–Verlag, Berlin, 2006, 253–282.
7. Ito, M., Martín–Vide, C., Păun, Gh.: A Characterization of Parikh Sets of ET0L Languages in Terms of P Systems. In Ito, M., Păun, Gh., Yu, S. (eds.): *Words, Semigroups, and Transductions*. World Scientific, 2001, 239–254.
8. von Koch, H.: Sur One Courbe Continue sans Tangente, Obtenue par une Construction Géometrique Élémentaire. *Arkiv för Matematik*, **1** (1904), 681–704.
9. von Koch, H.: Une Méthode Géometrique Élémentaire pour L'étude de Certaines Questions de la Théorie des Courbes Planes. *Acta Mathematica*, **30** (1906), 145–174.
10. Lindenmayer, A.: Mathematical Models for Cellular Interaction in Development, Parts I and II. *Journal of Theoretical Biology*, **18** (1968), 280–315.
11. Lindenmayer, A.: Developmental Systems without Cellular Interaction, Their Languages and Grammars. *Journal of Theoretical Biology,* **30** (1971), 455-484.

12. Madhu, M., Krithivasan, K.: P Systems with Membrane Creation: Universality and Efficiency. In Margenstern, M., Rogozhin, Y. (eds.): *Proceedings of the Third International Conference on Universal Machines and Computations*. Lecture Notes in Computer Science, Vol. 2055. Springer–Verlag, Berlin, 2001, 276–287.
13. Obtulowicz, A., Păun, Gh.: (In Search of) Probabilistic P Systems. *Biosystems*, **70**, 2 (2003), 107–121.
14. Papert, S.: *Mindstorms: Children, Computers and Powerful Ideas*. Basic Books, 1980.
15. Păun, Gh.: *Membrane Computing – An Introduction*. Springer-Verlag, Berlin, 2002.
16. Pescini, D., Besozzi, D., Mauri, G., Zandron, C.: Dynamical Probabilistic P Systems. *International Journal of Foundations of Computer Science*, **17**, 1 (2006), 183–204.
17. Prusinkiewicz, P., Lindenmayer, A., Hanan, J.: Developmental Models of Herbaceous Plants for Computer Imagery Purposes. *Computer Graphics*, **22**, 4 (1988), 141–150.
18. Prusinkiewicz, P.: Graphical Applications of L systems. In *Proceedings of Graphical Interface '86*. Kaufmann, 1986, 247–253.
19. Prusinkiewicz, P., Hanan, J.: *Lindenmayer Systems, Fractals and Plants*. Lecture Notes on Biomathematics, Vol. 79. Springer-Verlag, Berlin, 1989.
20. Prusinkiewicz, P., Lindenmayer, A.: *The Algorithmic Beauty of Plants*. Springer-Verlag, Berlin, 1990.
21. Romero–Jiménez, A., Gutiérrez–Naranjo, M.A., Pérez–Jiménez, M. J.: The Growth of Branching Structures with P Systems. In Graciani–Díaz, C., Păun, Gh., Romero–Jiménez, A., Sancho–Caparrini, F. (eds.): *Proceedings of the Fourth Brainstorming Week on Membrane Computing*, Vol. II. Fénix Editora, Sevilla, 2006, 253–265.
22. Salomaa, A.: *Formal Languages*. Academic Press (1973)
23. Smith, A.R.: Plants, Fractals and Formal Languages. *Computer Graphics*, **18**, 3 (1984), 1–10.
24. The P Systems webpage `http://psystems.disco.unimib.it/`

Identifying P Rules from Membrane Structures with an Error-Correcting Approach*

José M. Sempere and Damián López

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
{jsempere,dlopez}@dsic.upv.es

Abstract. In this work we propose an error-correcting approach to solve the identification of P rules for membrane modifications based on the behavior of the P system. To this aim, we take the framework of inductive inference from (structural) positive examples. The algorithm that we propose is based on previous definitions of distances between membrane structures and multiset tree automata.

1 Introduction

Membrane structures are linked to P systems as the structural information associated to them in order to make calculations. The membrane structure can be represented by a tree in which the internal nodes denote regions which have inner regions inside. The root of the tree is always associated to the skin membrane of the P system.

The relation between regions and trees has been strengthened by Freund *et al.* [6]. The authors established that any recursively enumerable set of trees can be generated by a P system with active membranes and string objects. In such a framework, P systems can be viewed as tree generators.

In this work we use multiset tree automata to accept and handle the tree structures defined by P systems [19] and we use edit distances between trees and multiset tree automata [11]. Edit distances for membrane systems were also considered in [4] and [5]. Here, we propose an inferring method to obtain a multiset tree automaton from a (finite) set of trees.

The structure of this work is as follows: First we introduce basic definitions and notation about multisets, tree languages, and automata and P systems. Then, we define multiset tree automata, we define the relation of *mirroring* between trees and we show some results between tree automata, multiset tree automata, and mirroring trees. We establish the minimum editing distance between membrane structures. In Section 3, we propose an error-correcting approach to infer multiset tree automata from examples of membrane structures. Finally, we mention some conclusions and give some guidelines for future works.

* Work supported by the Spanish CICYT under contract TIC2003-09319-C03-02 and the Generalitat Valenciana GV06/068.

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 507–520, 2006.

2 Notation and Definitions

In the sequel we will provide some concepts from formal language theory, membrane systems and multiset processing. We suggest the following books to the reader: [16], [14], and [2].

Multisets

First, we will provide some definitions from multiset theory as exposed in [20].

Definition 1. *Let D be a set. A multiset over D is a pair $\langle D, f\rangle$ where $f : D \longrightarrow \mathbb{N}$ is a function.*

Definition 2. *Let $A = \langle D, f\rangle$ be a multiset; we say that A is* empty *if for all $a \in D$, $f(a) = 0$.*

Definition 3. *Suppose that $A = \langle D, f\rangle$ and $B = \langle D, g\rangle$ are two multisets. Then*

1. *the removal of multiset B from A, denoted by $A \ominus B$, is the multiset $C = \langle D, h\rangle$ where for all $a \in D$ $h(a) = max(f(a) - g(a), 0)$,*
2. *their sum, denoted by $A \oplus B$, is the multiset $C = \langle D, h\rangle$, where for all $a \in D$ $h(a) = f(a) + g(a)$, and*
3. *we say that $A = B$ if the multiset $(A \ominus B) \oplus (B \ominus A)$ is empty.*

A multiset $M = \langle D, f\rangle$ whose size, $|M|$, defined by $\sum_{a\in D} f(a)$, is finite is said to be *bounded*. Formally, we will denote the set of all multisets $\langle D, f\rangle$ such that $\sum_{a\in D} f(a) = n$ by $\mathcal{M}_n(D)$.

A concept that is quite useful to work with sets and multisets is the *Parikh mapping*. Formally, a Parikh mapping can be viewed as the application $\Psi : D^* \rightarrow \mathbb{N}^n$ where $D = \{d_1, d_2, \cdots, d_n\}$. Given an element $x \in D^*$ we define $\Psi(x) = (\#_{d_1}(x), \cdots, \#_{d_n}(x))$ where $\#_{d_j}(x)$ denotes the number of occurrences of d_j in x.

P Systems

We introduce now some basic concepts from membrane systems taken from [14]. A general P system of degree m is a construct

$$\Pi = (V, T, C, \mu, w_1, \cdots, w_m, (R_1, \rho_1), \cdots, (R_m, \rho_m), i_0),$$

where:

- V is an alphabet (the *objects*)
- $T \subseteq V$ (the *output alphabet*)
- $C \subseteq V$, $C \cap T = \emptyset$ (the *catalysts*)
- μ is a membrane structure consisting of m membranes
- w_i, $1 \leq i \leq m$, is a string representing a multiset over V associated with the region i

- R_i, $1 \leq i \leq m$, is a finite set of *evolution rules* over V associated with the ith region and ρ_i is a partial order relation over R_i specifying a *priority*. An evolution rule is a pair (u, v) (or $u \rightarrow v$) where u is a string over V and $v = v'$ or $v = v'\delta$ where v' is a string over

$$\{a_{here}, a_{out}, a_{in_j} \mid a \in V, 1 \leq j \leq m\}$$

  and δ is an special symbol not in V (it defines the *membrane dissolving action*). From now on, we will denote the set $\{here, out, in_k \mid 1 \leq k \leq m\}$ by tar.
- i_0 is a number between 1 and m and it specifies the *output* membrane of Π (in the case that it equals to ∞ the output is read outside the system).

The language generated by Π in external mode ($i_0 = \infty$) is denoted by $L(\Pi)$ and it is defined as the set of strings that can be defined by collecting the objects that leave the system by arranging the leaving order (if several objects leave the system at the same time then permutations are allowed). The set of numbers that represent the objects in the output membrane i_0 will be denoted by $N(\Pi)$. Obviously, both sets $L(\Pi)$ and $N(\Pi)$ are defined only for *halting computations*.

One of the multiple variations of P systems is related to the creation, division, and modification of membrane structures. There have been several works in which these variants have been proposed (see, for example, [1,13,14,15]).

In the following, we enumerate some kinds of rules which are able to modify the membrane structure:

1. 2-division: $[_h a]_h \rightarrow [_{h'} b]_{h'} [_{h''} c]_{h''}$
2. Creation: $a \rightarrow [_h b]_h$
3. Dissolving: $[_h a]_h \rightarrow b$

The power of P systems with the previous operations and other ones (e.g., *exocytosis, endocytosis*, etc.) has been widely studied in the membrane computing area.

Tree Automata and Tree Languages

Now, we will introduce some concepts from tree languages and automata as exposed in [3,9]. First, let a *ranked alphabet* be the association of an alphabet V with a finite relation r in $V \times \mathbb{N}$. We denote by V_n the subset $\{\sigma \in V \mid (\sigma, n) \in r\}$. We will denote by $maxarity(V)$ the maximum integer n such that V_n is not empty.

The set V^T of trees over V, is defined inductively as follows:

$a \in V^T$ for every $a \in V_0$
$\sigma(t_1, \ldots, t_n) \in V^T$ whenever $\sigma \in V_n$ and $t_1, \ldots, t_n \in V^T$, $(n > 0)$

and let a *tree language* over V be defined as a subset of V^T.

Given the tuple $l = \langle 1, 2, \ldots, k \rangle$ we will denote the set of permutations of l by $perm(l)$. Let $t = \sigma(t_1, \ldots, t_n)$ be a tree over V^T, we will denote the set of

permutations of t at first level by $perm_1(t)$. Formally, $perm_1(t) = \{\sigma(t_{i_1}, \ldots, t_{i_n}) \mid \langle i_1, i_2, \ldots, i_n\rangle \in perm(\langle 1, 2, \ldots, n\rangle)\}$.

Let $\mathbb{N}^*$ be the set of finite strings of natural numbers, separated by dots, formed by using the catenation as the composition rule and the empty word λ as the identity. Let the prefix relation $\leq$ in $\mathbb{N}^*$ be defined by the condition that $u \leq v$ if and only if $u \cdot w = v$ for some $w \in \mathbb{N}^*$ $(u, v \in \mathbb{N}^*)$. A finite subset D of $\mathbb{N}^*$ is called a *tree domain* if:

$$u \leq v \text{ where } v \in D \text{ implies } u \in D, \text{ and}$$
$$u \cdot i \in D \text{ whenever } u \cdot j \in D \ (1 \leq i \leq j)$$

Each tree domain D could be seen as an unlabeled tree whose nodes correspond to the elements of D where the hierarchy relation is the prefix order. Thus, each tree t over V can be seen as an application $t : D \to V$. The set D is called the *domain of the tree* t, and denoted by $dom(t)$. The elements of the tree domain $dom(t)$ are called *positions* or *nodes* of the tree t. We denote by $t(x)$ the label of a given node x in $dom(t)$.

Let the level of $x \in dom(t)$ be $|x|$. Intuitively, the level of a node measures its distance from the root of the tree. Then, we can define the depth of a tree t as $depth(t) = max\{|x| \mid x \in dom(t)\}$. In the same way, for any tree t, we denote the size of the tree by $|t|$ and the set of subtrees of t (denoted with $Sub(t)$) as follows:

$$Sub(a) = \{a\} \text{ for all } a \in V_0$$
$$Sub(t) = \{t\} \cup \bigcup_{i=1,\ldots,n} Sub(t_i) \text{ for } t = \sigma(t_1, \ldots, t_n) \ (n > 0)$$

Given a tree $t = \sigma(t_1, \ldots, t_n)$, the root of t will be denoted as $root(t)$ and defined as $root(t) = \sigma$. If $t = a$ then $root(t) = a$. The successors of a tree $t = \sigma(t_1, \ldots, t_n)$ will be defined as $H^t = \langle root(t_1), \ldots, root(t_n)\rangle$. Finally, $leaves(t)$ will denote the set of leaves of the tree t.

Definition 4. *A* finite deterministic tree automaton *is defined by the tuple* $A = (Q, V, \delta, F)$ *where* Q *is a finite set of states,* V *is a ranked alphabet,* $Q \cap V = \emptyset$, $F \subseteq Q$ *is the set of final states and* $\delta = \bigcup_{i:V_i \neq \emptyset} \delta_i$ *is a set of transitions defined as follows:*

$$\delta_n : (V_n \times (Q \cup V_0)^n) \to Q \qquad n = 1, \ldots, m$$
$$\delta_0(a) = a \qquad \forall a \in V_0$$

Given the state $q \in Q$, we define the *ancestors* of the state q, denoted by $Ant(q)$, as the set of strings

$$Ant(q) = \{p_1 \cdots p_n \mid p_i \in Q \cup V_0 \wedge \delta_n(\sigma, p_1, \ldots, p_n) = q\}$$

From now on, we will refer to finite deterministic tree automata simply as *tree automata*. We suggest [3,9] for other definitions on tree automata.

The transition function δ is extended to a function $\delta : V^T \to Q \cup V_0$ on trees as follows:

$$\delta(a) = a \text{ for any } a \in V_0$$
$$\delta(t) = \delta_n(\sigma, \delta(t_1), \ldots, \delta(t_n)) \text{ for } t = \sigma(t_1, \ldots, t_n) \ (n > 0)$$

Note that the symbol δ denotes both the set of transition functions of the automaton and the extension of these functions to operate on trees. In addition, you can observe that the tree automaton A cannot accept any tree of depth zero.

Given a finite set of trees T, let the *subtree automaton* for T be defined as $AB_T = (Q, V, \delta, F)$, where:

$$Q = Sub(T)$$
$$F = T$$
$$\delta_n(\sigma, u_1, \ldots, u_n) = \sigma(u_1, \ldots, u_n) \qquad \sigma(u_1, \ldots, u_n) \in Q$$
$$\delta_0(a) = a \qquad a \in V_0$$

Multiset Tree Automata and Mirrored Trees

We will extend over multisets some definitions of tree automata and tree languages. We will introduce the concept of multiset tree automata and then we will characterize the set of trees that it accepts.

Given any tree automaton $A = (Q, V, \delta, F)$ and $\delta_n(\sigma, p_1, p_2, \ldots, p_n) \in \delta$, we can associate to δ_n the multiset $\langle Q \cup V_0, f\rangle \in \mathcal{M}_n(Q \cup V_0)$ where f is defined by $\Psi(p_1 p_2 \ldots p_n)$. The multiset defined in such way will be denoted by $M_\Psi(\delta_n)$. Alternatively, we can define $M_\Psi(\delta_n)$ as $M_\Psi(p_1) \oplus M_\Psi(p_2) \oplus \cdots \oplus M_\Psi(p_n)$ where $\forall 1 \leq i \leq n \ M_\Psi(p_i) \in \mathcal{M}_1(Q \cup V_0)$. Observe that if $\delta_n(\sigma, p_1, p_2, \ldots, p_n) \in \delta$, $\delta'_n(\sigma, p'_1, p'_2, \ldots, p'_n) \in \delta$ and $M_\Psi(\delta_n) = M_\Psi(\delta'_n)$ then δ_n and δ'_n are defined over the same set of states and symbols but in different order (that is the multiset induced by $\langle p_1, p_2, \cdots, p_n\rangle$ equals to the one induced by $\langle p'_1 p'_2 \ldots p'_n\rangle$).

Now, we can define a *multiset tree automaton* that performs a bottom-up parsing as in the tree automaton case.

Definition 5. *A* multiset tree automaton *is a tuple $MA = (Q, V, \delta, F)$, where Q is a finite set of states, V is a ranked alphabet with maxarity$(V) = n$, $Q \cap V = \emptyset$, $F \subseteq Q$ is a set of final states and δ is a set of transitions defined as follows:*

$$\delta = \bigcup_{\substack{1 \leq i \leq n \\ i : V_i \neq \emptyset}} \delta_i$$

$$\delta_i : (V_i \times \mathcal{M}_i(Q \cup V_0)) \to \mathcal{P}(\mathcal{M}_1(Q)) \qquad i = 1, \ldots, n$$
$$\delta_0(a) = M_\Psi(a) \in \mathcal{M}_1(Q \cup V_0) \qquad \forall a \in V_0$$

We can take notice that every tree automaton A defines a multiset tree automaton MA as follows.

Definition 6. *Let $A = (Q, V, \delta, F)$ be a tree automaton. The multiset tree automaton induced by A is defined by the tuple $MA = (Q, V, \delta', F)$ where each δ' is defined as follows: $M_\Psi(r) \in \delta'_n(\sigma, M)$ if $\delta_n(\sigma, p_1, p_2, \ldots, p_n) = r$ and $M_\Psi(\delta_n) = M$.*

Observe that, in the general case, the multiset tree automaton induced by A is nondeterministic.

As in the case of tree automata, δ' could also be extended to operate on trees. Here, the automaton carries out a bottom-up parsing where the tuples of states and/or symbols are transformed by using the Parikh mapping Ψ to obtain the multisets in $\mathcal{M}_n(Q \cup V_0)$. If the analysis is completed and δ' returns a multiset with at least one final state, then the input tree is accepted. So, δ' can be extended as follows:

$$\delta'(a) = M_\Psi(a) \text{ for any } a \in V_0,$$

$$\delta'(t) = \{M \in \delta'_n(\sigma, M_1 \oplus \cdots \oplus M_n) \mid M_i \in \delta'(t_i)\ 1 \leq i \leq n\} \text{ for } t = \sigma(t_1, \ldots, t_n)\ (n > 0).$$

Formally, every multiset tree automaton MA accepts the following language

$$L(MA) = \{t \in V^T \mid M_\Psi(q) \in \delta'(t), q \in F\}.$$

Another extension which will be useful is the one related to the ancestors of every state. So, we define $Ant_\Psi(q) = \{M \mid M_\Psi(q) \in \delta_n(\sigma, M)\}$.

Theorem 1. *(Sempere and López, [19]) Let $A = (Q, V, \delta, F)$ be a tree automaton, $MA = (Q, V, \delta', F)$ be the multiset tree automaton induced by A and $t = \sigma(t_1, \ldots, t_n) \in V^T$. If $\delta(t) = q$, then $M_\Psi(q) \in \delta'(t)$.*

Corollary 1. *(Sempere and López, [19]) Let $A = (Q, V, \delta, F)$ be a tree automaton and $MA = (Q, V, \delta', F)$ be the multiset tree automaton induced by A. If $t \in L(A)$, then $t \in L(MA)$.*

We will introduce the concept of *mirroring* in tree structures as exposed in [19]. Informally speaking, two trees will be related by mirroring if some permutations at the structural level hold. We propose a definition that relates all the trees with this mirroring property.

Definition 7. *Let t and s be two trees from V^T. We will say that t and s are* mirror equivalent*, denoted by $t \bowtie s$, if one of the following conditions holds:*

1. $t = s = a \in V_0$,
2. $t \in perm_1(s)$,
3. $t = \sigma(t_1, \ldots, t_n)$, $s = \sigma(s_1, \ldots, s_n)$ *and there exists* $< s^1, s^2, \ldots, s^k > \in perm(< s_1, s_2, \ldots, s_n >)$ *such that* $\forall 1 \leq i \leq n\ t_i \bowtie s^i$.

Theorem 2. *(Sempere and López, [19]) Let $A = (Q, V, \delta, F)$ be a tree automaton, $t = \sigma(t_1, \ldots, t_n) \in V^T$ and $s = \sigma(s_1, \ldots, s_n) \in V^T$. Let $MA = (Q, V, \delta', F)$ be the multiset tree automaton induced by A. If $t \bowtie s$, then $\delta'(t) = \delta'(s)$.*

Corollary 2. *(Sempere and López, [19]) Let $A = (Q, V, \delta, F)$ be a tree automaton, $MA = (Q, V, \delta', F)$ the multiset tree automaton induced by A and $t \in V^T$. If $t \in L(MA)$, then, for any $s \in V^T$ such that $t \bowtie s$, $s \in L(MA)$.*

The last results were useful to propose an algorithm to determine whether two trees are mirror equivalent or not [19]. So, given two trees s and t, we can establish in time $\mathcal{O}((min\{|t|, |s|\})^2)$ whether or not $t \bowtie s$.

Editing Distances Between Membrane Structures

The initial order of a membrane structure can be fixed. Anyway, whenever the system evolves (membrane dissolving, division, creation, etc.) this order can be at least somehow ambiguous. Furthermore, the initial order of a P system is only a naming convention given that the membrane structure of any P system can be renamed without changing its behavior due to the parallelism ingredient (observe that if this mechanism was sequential, then the ordering could be important for the final output).

The representation by trees could be essential for the analysis of the dynamic behavior of P systems. Whenever we work with trees to represent the membrane structure of a given P system, we can find a *mirroring effect*.

Example 1. Let us take the three membrane structures of Figure 1. Then, the associated trees are the following (in top-down order):

$$\sigma(\sigma(a, a), \sigma(a, \sigma(a)))$$
$$\sigma(\sigma(\sigma(a), \sigma(a, a)), \sigma(a, a))$$
$$\sigma(\sigma(\sigma(a, \sigma(a), a), a), \sigma(a, a))$$

Observe that the symbol a denotes the names for elementary regions instead of objects.

We proposed a method to establish if two membrane structures μ and $\mu\prime$ are identical [19], while in [11] we proposed an algorithm to obtain the minimum set of membrane rule applications needed to transform μ into $\mu\prime$ (or vice versa). The last algorithm was based on multiset tree automata and the tree representation for membrane structures. Note that the target of that algorithm was to force the automaton to accept the tree. The algorithm employed edit operations for substitution (reduction) of a tree to a state of the automaton, deletion of a (sub)tree and insertion of a state. Intuitively, the substitution of a tree by a state of the automaton could be seen as the substitution of the tree by the nearest tree that could be reduced to the state.

3 Identification with an Error-Correcting Approach

Here, we address the problem of inferring some operators that regulate the behavior of a P system: given a set of membrane structures generated by an arbitrary

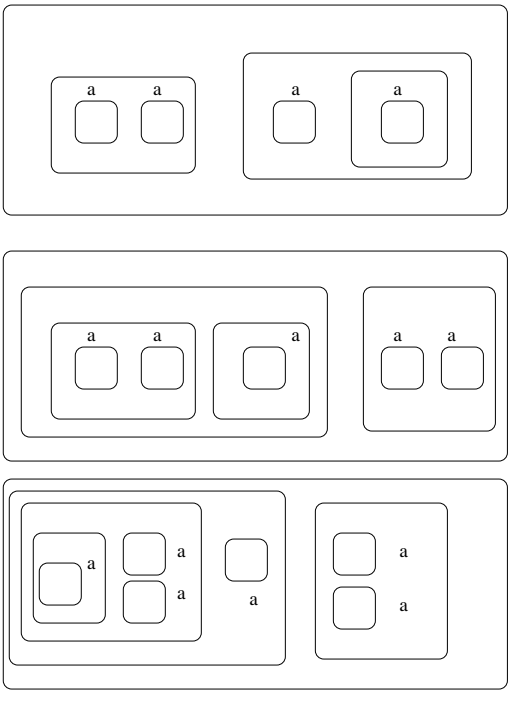


Fig. 1. Three membrane structures

P system, the problem consists of defining the set of rules needed to generate the membrane structures of the set and (possibly) some others that do not belong to it.

This problem could be approached as a *Grammatical Inference* problem [18]. Here, the set of membrane structures can be represented by a set of trees and the set of rules that regulate the P system behavior is deduced from a multiset tree automata. So, we will solve the problem by using inference methods for tree languages.

The problem of inferring tree languages has been widely approached in the grammatical inference literature. So, in [7] a method to infer k-testable tree sets from sample data is proposed. In [8] a method to infer recognizable tree languages from finite information is proposed. Finally, in [12] a method to infer reversible tree languages from samples was proposed.

Here, we will use an inferring method based on error-correcting techniques. The method that we propose is based on a preliminary technique for tree automata inference [10] based on the *Error-Correcting Grammatical Inference* method (ECGI) [17]. Basically, the method constructs the set of transitions of the automaton such that the editing distances from the sample data to the automaton is minimal.

In what follows, we propose an adaptation of such method. We will use multiset tree automata instead of tree automata and the editing operations together with the minimal editing distance is based on our previous work [11]. So, we propose Algorithm 1 as a method to infer multiset tree automata from a finite

sample of trees. Algorithm 1 uses a subroutine showed as Algorithm 2 which calculates the minimal editing distance for every tree to the current multiset tree automaton and adds the new transitions needed to converge to the minimal cost tree automaton.

Algorithm 1. Error correcting multiset tree automata inference algorithm.

Input:

– A set of trees $S = \{t_1, t_2, \ldots t_n\}$ (Membrane representations)

Output:

– A multiset tree automaton $A = (Q, V, \delta, F)$ that, at least, recognizes the set of trees $\{s : \exists t \in S,\ t \bowtie s\}$

Method:

1. Obtain the initial automaton
   – $V = \{\sigma\} \cup leaves(t_1)$
   – $\forall s \in Sub(t_1)$ in postorder
     if $s = \sigma(u_1, u_2, \ldots, u_p)$ then $Q = Q \cup \{< u_1 u_2 \ldots u_p >\}$
     else /*$s \in leaves(t_1)$*/ $Q = Q \cup s$
     End$\forall$
   – $F = \{< u_1 u_2 \ldots u_k >\}$ where $t_1 = \sigma(u_1, u_2, \ldots, u_k)$
   – $\delta(a) = a\ \forall a \in leaves(t_1)$
     if $u_1, u_2, \ldots u_k, < u_1 u_2 \ldots u_k > \in Q$ then add to the automaton
     the transition $\delta(\sigma, u_1 u_2 \ldots u_k) =< u_1 u_2 \ldots u_k >$
2. $\forall t_i \in S$ with $2 \leq i \leq n$
   – $A = Expand(A, t_i)$
   End$\forall$
3. $Return(A)$

EndMethod.

The output of the Algorithm 1 is a multiset tree automaton that recognizes all the trees given as input at the minimum editing cost together with the mirrored trees of the input. Observe, that this automaton could recognize some other trees that have not been supplied as input. The correctness of the algorithm and its complexity, which is polynomial with the size of the tree set, is deduced from our previous work on editing distance [11].

We will discuss a complete example of the algorithm running in order to clarify its behavior.

Example 2. Let us consider the following training set:

$$\left\{ \begin{array}{c} \sigma(\sigma(a,a),\sigma(a,\sigma(a))), \\ \sigma(\sigma(\sigma(a),\sigma(a,a)),\sigma(a,a)), \\ \sigma(\sigma(\sigma(a,\sigma(a),a),a),\sigma(a,a)) \end{array} \right\}$$

which corresponds to the membrane structures showed in Figure 1.

Algorithm 2. Automaton modification subroutine $Expand(A, t)$.

Input:

- A multiset tree automaton $A = (Q, V, \delta, \{q_f\})$ and a tree $t = \sigma(t_1, t_2, \ldots, t_m)$ (membrane representation)

Output:

- A multiset tree automaton $A = (Q, V, \delta, F)$ that accepts, at least, $L(A) \cup \{s : s \bowtie t\}$

Method:

1. Consider the input automaton A and perform an error correcting analysis for t according to [11]
2. From the previous step, obtain the accepting minimum cost path Δ_t. Distinguish those non-error transitions Δ_t^N from the error transitions Δ_t^E
3. $\forall s \in Sub(t)$
   if $s = (\sigma, s_1, \ldots, s_k)$ then
   consider τ_s the transition $\delta(\sigma, q_1 \ldots q_k) = p_s \in \Delta_t$
   and set $Red[s] = p_s$
   else /* $s \in leaves(t)$ */ set $Red[s] = s$
   End$\forall$
4. $\forall s = \sigma(s_1, \ldots, s_k) \in Sub(t)$ /* in postorder */
   if $\tau_s \in \Delta_t^E$ or $Red[s] = \emptyset$ then
   add a new state q_N to Q
   add the transition $\delta(\sigma, Red[s_1] \ldots Red[s_k]) = q_N$
   set $Red[s] = q_N$
   else if $\exists \tau_{s_i} \in \Delta_t^E$ then add the transition $\delta(\sigma, Red[s_1] \ldots Red[s_k]) = q_N$
   End$\forall$
5. add the transition $\delta(\sigma, Red[t_1] Red[t_2] \ldots Red[t_m]) = q_f$
6. $Return(A)$

EndMethod.

The first step in Algorithm 1 considers the empty automaton and the first tree, thus obtaining the initial multiset tree automaton with the following transitions:

$$\begin{array}{r} \delta(\sigma, aa) = q_1 \\ \delta(\sigma, a) = q_2 \\ \delta(\sigma, aq_2) = q_3 \\ \delta(\sigma, q_1 q_3) = q_4 \in F \end{array}$$

The second inference step performs an error correcting analysis of the second tree in the training set and the previous automaton. The distance matrix is shown in Table 1. Figure 2 summarizes the error correcting analysis.

Table 1. Distance table of each subtree (in postorder) and each state of the automaton. Minimum cost path is boldmarked. Second postorder subtree is deleted in the analysis, therefore its row contains no bold figure. Note that it is not necessary to obtain all the distances for the root node because only distances to final states will be considered.

	q_1	q_2	q_3	q_4
s_1	1	**0**	2	8
s_2	0	1	3	9
s_3	7	6	**4**	2
s_3	**0**	1	3	7
s_5	–	–	–	**4**

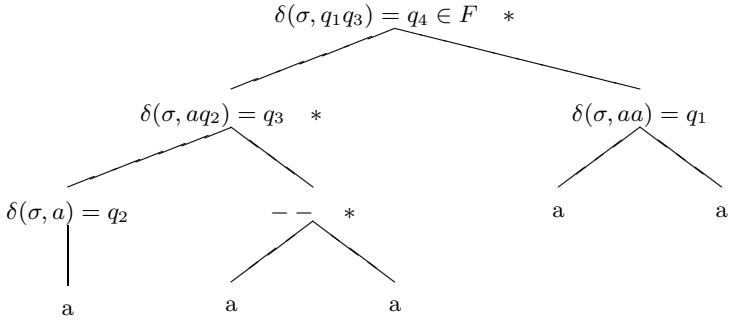


Fig. 2. Error correcting parsing of the tree $\sigma(\sigma(\sigma(a), \sigma(a, a)), \sigma(a, a))$. Error productions are marked with an asterisk. Note that a subtree deletion also occurred. This edit operation is also considered as an error transition.

The error correcting analysis obtains three error transitions. The automaton is then modified to accept the sample. Three new transitions are added (marked with an asterisk) in the following way:

$$\begin{aligned}
\delta(\sigma, aa) &= q_1 \\
\delta(\sigma, a) &= q_2 \\
\delta(\sigma, aq_2) &= q_3 \\
\delta(\sigma, q_1q_3) &= q_4 \in F \\
(*)\quad \delta(\sigma, aa) &= q_5 \\
(*)\quad \delta(\sigma, q_2q_5) &= q_6 \\
(*)\quad \delta(\sigma, q_6q_1) &= q_4 \in F
\end{aligned}$$

Note that the inference process adds a transition which is equal to an already exiting one. This case can be run-time detected, using the existing transition instead of creating a new one. This automaton is considered in the following inference step. This step considers the following tree

$$\sigma(\sigma(\sigma(a, \sigma(a), a), a), \sigma(a, a))$$

The distance matrix is shown in Table 2. Figure 3 summarizes the error correcting analysis.

Table 2. Distance table of each subtree (in postorder) and each state of the automaton. Minimum cost path is boldmarked.

	q_1	q_2	q_3	q_4	q_5	q_6
s_1	1	0	2	8	1	6
s_2	2	**3**	1	7	2	5
s_3	6	5	**4**	6	6	6
s_3	**0**	1	3	9	0	7
s_5	–	–	–	–	–	**4**

The final automaton is the following:

$$
\begin{array}{rr}
 & \delta(\sigma, aa) = q_1 \\
 & \delta(\sigma, a) = q_2 \\
 & \delta(\sigma, aq_2) = q_3 \\
 & \delta(\sigma, q_1q_3) = q_4 \in F \\
 & \delta(\sigma, aa) = q_5 \\
 & \delta(\sigma, q_2q_5) = q_6 \\
 & \delta(\sigma, q_6q_1) = q_4 \in F \\
(*) & \delta(\sigma, a) = q_7 \\
(*) & \delta(\sigma, aq_7a) = q_8 \\
(*) & \delta(\sigma, aq_8) = q_3
\end{array}
$$

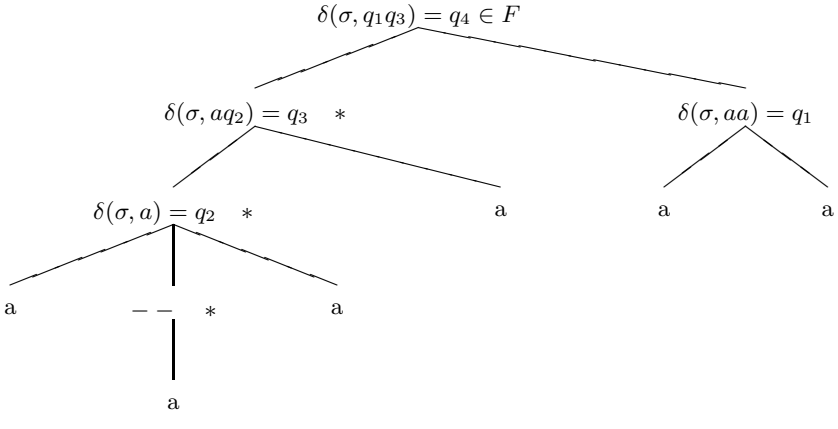


Fig. 3. Error correcting parsing of the tree $\sigma(\sigma(a, \sigma(a, \sigma(a), a), a), \sigma(a, a))$. Error productions are marked with an asterisk. Note that the transition used to reduce the third postorder subtree is marked as error. This transition does not add error cost but one of its descendants is an error transition and therefore a new transition must be created.

In the last example we obtain a multiset tree automaton which is consistent with the input data. Observe that this automaton induces a set of P rules for a P system as was showed in [19] and [11].

4 Conclusions and Future Work

We have proposed a method to infer a multiset tree automaton from a finite set of membrane structures. This multiset tree automaton employs the minimum number of tree editing operations to accept the input set. From this multiset tree automaton we can obtain a set of P rules that regulates the behavior of a P system which generates the membrane structures given as input.

Here, we have used a grammatical inference technique based on an error-correcting approach. In the future we will explore other options to infer new models (i.e., reversible tree languages, locally testable tree languages, etc.) So, a new question arises: Given a set of trees of a tree language class (reversible, locally testable, etc.), what is the common characteristics of the P systems that generate them? Such a characteristic will introduce a characterization of P systems based on the kind of membrane structures that they generate.

References

1. A. Alhazov, T.O. Ishdorj. Membrane operations in P systems with active membranes. In *Proc. Second Brainstorming Week on Membrane Computing*, TR 01/04 of RGNC, Sevilla University, 2004, 37–44.
2. C. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, *Multiset Processing*. LNCS 2235, Springer, Berlin, 2001.
3. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi. *Tree automata techniques and applications.* Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997, release October, 1rst 2002.
4. A. Cordón-Franco, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez. Weak metrics on configurations of a P system. In *Proc. Second Brainstorming Week on Membrane Computing*, RGNC TR 01/04 of RGNC, Sevilla University, 2004, 139–151.
5. E. Csuhaj-Varjú, A. Di Nola, Gh. Păun, M.J. Pérez-Jiménez, G. Vaszil. Editing configurations of P systems. In *3rd Brainstorming Week on Membrane Computing 2005*, TR 01/05 of RGNC, Sevilla University (M.A. Gutiérrez Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan, eds.), Fénix Editora, Sevilla, 2005, 131–155.
6. R. Freund, M. Oswald, A. Păun. P systems generating trees. In *Pre-proceedings of Fifth Workshop on Membrane Computing (WMC5)* (G. Mauri, Gh. Păun, C. Zandron, eds.), MolCoNet project IST-2001-32008, 2004, 221–232.
7. P. García. *Learning k-Testable Tree Sets from Positive Data.* Informe técnico DSIC-II/46/93, 1993.
8. P. García, J. Oncina. *Inference of Recognizable Tree Sets.* Informe técnico DSIC-II/47/93, 1993.
9. F. Gécseg, M. Steinby. Tree languages. In *Handbook of Formal Languages*, volume 3 (G. Rozenberg, A. Salomaa, eds.), Springer-Verlag, Berlin, 1997, 1–69.

10. D. López, S. España. Error correcting tree language inference. *Pattern Recognition Letters*, 23, 1-3 (2002), 1–12.
11. D. López, J.M. Sempere. Editing distances between membrane structures. In *Proceedings of the 6th International Workshop, WMC 2005* (R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 3850, Springer, Berlin, 2006, 326–341.
12. D. López, J.M. Sempere, P. García Inference of reversible tree languages. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, 34, 4 (2004), 1658–1665.
13. A. Păun. On P systems with active membranes. In *Proc. of the First Conference on Unconventionals Models of Computation*, 2000, 187–201.
14. Gh. Păun. *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
15. Gh. Păun, Y. Suzuki, H. Tanaka, T. Yokomori. On the power of membrane division on P systems. *Theoretical Computer Science* 324, 1 (2004), 61–85.
16. G. Rozenberg, A. Salomaa, eds.. *Handbook of Formal Languages* Springer, Berlin, 1997.
17. H. M. Rulot *ECGI. Un algoritmo de inferencia gramatical mediante corrección de errores.* PhD Dissertation, Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia 1992 (in Spanish).
18. Y. Sakakibara. Recent advances of grammatical inference. *Theoretical Computer Science*, 185 (1997), 15–45.
19. J.M. Sempere, D. López. Recognizing membrane structures with tree automata. In *3rd Brainstorming Week on Membrane Computing 2005*, TR 01/05 of RGNC, Sevilla University (M.A. Gutiérrez Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan, eds.), Fénix Editora, Sevilla, 2005, 305–316.
20. A. Syropoulos. Mathematics of multisets. In [2], 347–358.

Computational Completeness of Tissue P Systems with Conditional Uniport

Sergey Verlan[1], Francesco Bernardini[2], Marian Gheorghe[3], and Maurice Margenstern[4]

[1] LACL, Département Informatique
Université Paris 12
61 av. Général de Gaulle, 94010 Crétiel, France
verlan@univ-paris12.fr
[2] Leiden Institute of Advanced Computer Science
Universiteit Leiden
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
bernardi@liacs.nl
[3] Department of Computer Science
The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
M.Gheorghe@dcs.shef.ac.uk
[4] Université Paul Verlaine - Metz, UFR MIM, LITA, EA 3097
Ile du Saulcy, 57045 Metz Cédex, France
margens@univ-metz.fr

Abstract. The paper introduces (purely communicative) tissue P systems with conditional uniport. Conditional uniport means that rules move only one object at a time, but this may be with the help of another one acting as an activator which is left untouched in the place where it is. Tissue P systems with conditional uniport are shown to be computationally complete in the sense that they can recognize all recursively enumerable sets of natural numbers. This is achieved by simulating deterministic register machines.

1 Introduction

Membrane computing is an emerging branch of natural computing which deals with distributed and parallel computing devices of a bio-inspired type, which are called *membrane systems* or *P systems* (see [13], [14], and also [19] for a comprehensive bibliography of P systems). P systems, originally devised by Gh. Păun in [13], are introduced as computing devices which abstract from the structure and functioning of living cells – they are defined as a hierarchical arrangement of regions delimited by membranes (*membrane structure*), with each region having associated a multiset of objects and a finite set of rules. Communication of objects through membranes is one of the fundamental features of every membrane system and this naturally led to the question of closely investigating the power of communication, that is, considering *purely communicative P systems* where objects are never modified but they just change their place within the system.

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 521–535, 2006.

A first attempt to address this issue was done in [10] by considering certain "vehicle-objects" (an abstraction for plasmids or for vectors from gene cloning) which actively carry objects through membranes. Then, the bio-chemical idea of membrane transport of pairs of molecules was considered in [12] by introducing the notion of *P systems with symport/antiport*. In such P systems, multisets of objects, here denoted by x, y, are moved across the membranes by means of rules of the form (x, in), (x, out) (*symport rules*, where multiset x is moved in one direction being either from outside to inside a membrane or from inside to outside a membrane, respectively), and $(x, out; y, in)$ (*antiport rules*, where multiset x is moved from inside to outside a membrane and multiset y is simultaneously moved from outside to inside a membrane); x, y can be of any size but they cannot be empty. As it happens in few other models (e.g., see the billiard ball model [18], the chip firing game [8], or railway simulation [9]), computing by communication turns out to be computationally complete: by using only symport and antiport rules, we can generate all Turing computable sets of numbers [12]. However, as in the aforementioned models of other inspiration, in order to generate all Turing computable sets, some infinity must be present and, in P systems with symport/antiport this is provided in the form of an infinite supply of objects taken from an external environment. Several subsequent works have been dedicated to improve this result in what concerns both the number of membranes used and the size of symport/antiport rules used inside the membranes. We refer to [16] for a survey of these investigations.

The class of membrane computing models was later extended to *tissue P systems* [14]: a variant of P systems where the underlying structure is defined as an arbitrary graph that is introduced as an abstraction for the organization in tissues of cells in multicellular organisms. Nodes in the graph represent *cells* (i.e., elementary membranes) that are able to communicate objects alongside the edges of this graph [14]. In particular, purely communicative tissue P systems with symport/antiport were investigated in [14] by showing completeness results for systems using rules of different sizes and different structures for the underlying graph. More recently, it was proved in [2] that tissue P systems with symport/antiport rules of a minimal size (i.e., rules of the forms (a, in), (a, out), $(a, out; b, in)$, with a, b objects from a given alphabet) are computational complete and two cells suffice.

In this paper we consider tissue P systems with a different model of communication called *conditional uniport*. Conditional uniport means that every application of a communication rule moves one object in a certain direction by possibly using another one as an activator which is left untouched in the place where it is. In other words, rules are assigned to the edges of the graph and they represent channels of communication; in the cell placed at one end of an edge, an object is used to activate the channel and either receives another object (in a single copy) from the cell at the other end of that edge, or sends another object (in one single copy) to the cell at the other end of the edge. The biological motivation for conditional uniport is twofold. On the one hand, in living cells, the active transport of small molecules is driven by protein channels: a molecule binds to one of

these channels which, through a change of conformation, is able to release the molecule outside the compartment (see [1]). On the other hand, in tissues, cell-to-cell communication depends heavily on extracellular signal molecules, which are produced by a cell to signal their neighbors or cells that are further away. In turn, these cells can respond to extracellular signal molecules by means of particular proteins called receptors; each receptor binds at cell-surface level to particular signal molecules and it is able to initiate a specific response inside the cell (see [1]). In both cases, it is only the small molecule or the signal molecule to move in one direction and, in order to be transported or to be recognized, it requires another object, a protein channel or a receptor. Such a model of communication was already investigated in [3] where an evolution-communication model was considered. This means that, besides conditional uniport, multiset rewriting rules can be used to modify the objects placed inside the cells. The main result reported in [3] showed how to achieve computational completeness by having only 2 cells and using non-cooperative multiset rewriting rules. Here we instead focus on the purely communicative case, with the usual assumption of an infinite supply of objects from the environment, and we show that tissue P systems with conditional uniport are able to simulate deterministic register machines by using 24 cells. This means that they can recognize all recursively enumerable sets of natural numbers.

2 Preliminaries

We recall here some basic notions concerning the notation commonly used in membrane computing and the few notions of formal language theory we need in the rest of the paper. We refer to [14], [17] for further details.

An alphabet is a finite non-empty set of abstract symbols. Given an alphabet V, we denote by V^* the set of all possible strings over V, including the empty string λ. The length of a string $x \in V^*$ is denoted by $|x|$ and, for each $a \in V$, $|x|_a$ denotes the number of occurrences of the symbol a in x. A multiset over V is a mapping $M : V \longrightarrow \mathbb{N}$ such that, $M(a)$ defines the multiplicity of a in the multiset M ($\mathbb{N}$ denotes the set of natural numbers). Such a multiset can be represented by a string $a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)} \in V^*$ and by all its permutations, with $a_j \in V$, $M(a_j) \neq 0$, $1 \leq j \leq n$. In other words, each string $x \in V^*$ identifies a multiset over V defined by $M_x = \{ (a, |x|_a) \mid a \in V \}$.

We also recall the notion of a (deterministic) *register machine* [11]. A deterministic *register machine* is the following construction:

$$M = (Q, R, q_0, q_f, P),$$

where Q is a set of states, $R = \{A_1, \dots, A_k\}$ is the set of registers, $q_0 \in Q$ is the initial state, $q_f \in Q$ is the final state and P is a set of instructions (called also rules) of the following form:

1. $(p, A+, q) \in P$, $p, q \in Q, p \neq q, A \in R$ (being in state p, increase register A and go to state q).

2. $(p, A-, q, s) \in P$, $p, q, s \in Q, A \in R$ (being in state p, decrease register A and go to q if successful or to s if A is zero).
3. $(q_f, STOP)$.

Moreover, for each state $p \in Q$, there is only one instruction of one of the above types.

A configuration of a register machine is given by the $k+1$-tuple $(q, n_1, \ldots, n_k)$ describing the current state of the machine as well as the contents of all registers. A transition of the register machine consists in updating/checking the value of a register according to an instruction of one of types above and by changing the current state to another one. We say that M computes a value $y \in \mathbb{N}$ on the *input* $x \in \mathbb{N}$ if starting from the initial configuration $(q_0, x, 0, \ldots, 0)$ it reaches the final configuration $(q_f, y, 0, \ldots, 0)$. We say that M recognizes the set $S \subseteq \mathbb{N}$ if for any input $x \in S$ the machine stops and for any $y \notin S$ the machine performs an infinite computation. It is known that register machines recognize all recursively enumerable sets of numbers [11].

We may also consider non-deterministic register machines where the first type of instruction is of the form $(p, A+, q, s)$ and with the following meaning: if the machine is in state p, then the register A is increased and the current state is changed to q or s non-deterministically. In this case the result of the computation is the set of all values of the first register when, starting with the configuration $(q_0, 0, 0, \ldots, 0)$, the computation eventually halts. We assume that the machine empties all registers except the first register before stopping. It is known that non-deterministic register machines generate all recursively enumerable sets of non-negative natural numbers starting from empty registers [11].

3 The Model

Now we formally introduce the notion of tissue P systems with conditional uniport.

Definition 1. *A* tissue P systems with conditional uniport *(a* TPCU*, for short) of degree $n \geq 1$ is a construct*

$$\mathcal{T} = (V, E, w_1, \ldots, w_n, R, c_I)$$

where:

1. *V is a* finite alphabet*;*
2. *$E \subseteq V$ is the* set of symbols which appear in the environment*;*
3. *$w_i \in V^*$, for all $1 \leq i \leq n$, is the multiset* initially associated to cell *i;*
4. *R is a finite set of* rules *of the forms:*
   (a) *$(i, a \to j)$ with $1 \leq i, j \leq n$, $i \neq j$ and $a \in V$,*
   (b) *$(i, a \to j, b)$ with $1 \leq i, j \leq n$, $i \neq j$ and $a, b \in V$,*
   (c) *$(i, b, a \to j)$ with $1 \leq i, j \leq n$, $i \neq j$ and $a, b \in V$,*
   (d) *$(0, a \to j, b)$ with $1 \leq j \leq n$ and $a, b \in V$,*
   (e) *$(i, b, a \to 0)$ with $1 \leq i \leq n$ and $a, b \in V$,*

(f) $(i, a \to 0)$ *with* $1 \leq i \leq n$, *and* $a \in V$,
(g) $(0, a \to i)$ *with* $1 \leq i \leq n$, *and* $a \in V$;

5. $c_I \in \{1, \ldots, n\}$ *is the* input cell.

Thus, a TPCU is defined as a collection of $n \geq 1$ cells denoted by $1, 2, \ldots, n$, each one of them containing a multiset over the given alphabet V. We also consider that the environment contains infinitely many copies of the objects in E and initially it does not contain any object in $V \setminus E$. Cells are allowed to interact with each other and with the environment through the application of the rules in R. A rule $(i, a \to j)$ specifies that an object a may be moved from cell i to cell j without any condition. A rule $(i, a \to j, b)$ with $1 \leq i, j \leq n$, $i \neq j$ and $a, b \in V$, specifies that if an object b is in cell j, then an occurrence of symbol a may be moved from cell i to cell j. A rule $(i, b, a \to j)$, with $1 \leq i, j \leq n$, $i \neq j$ and $a, b \in V$, specifies that if an object a and an object b are both present inside cell i, then that object a may be moved from cell i to cell j. Similarly, in presence of an object b inside cell j, a rule $(0, a \to j, b)$ may be used to move an occurrence of object a from the environment, denoted by 0, to cell j; if an object a and an object b are both present inside cell i, then a rule $(i, b, a \to 0)$ may be used to move that object a from cell i to the environment. Rules $(i, a \to 0)$, $(i, a \to 0)$ can instead be used to move an object a from cell i to the environment and vice versa without any condition.

As usual in membrane computing, we adopt a non-deterministic maximally parallel strategy for the application of the rules which makes the system transit from one configuration to the other. Specifically, we have that, in each step, all the rules that can be applied, depending on the current distribution of objects inside the cells, must be applied. Objects are non-deterministically assigned to the rules with the only restriction that, within the same step, the same occurrence of the same symbol is used by at most one rule. This means that an occurrence of a symbol cannot be simultaneously moved and used to move another object, and can be used to move at most one occurrence of another symbol. However, the same rule can be used in parallel many different times to move many different objects. Moreover, notice that rules do not modify the objects involved in their applications, hence, whenever a rule involves two objects, one is moved into some other cell whereas the other one is left untouched in the place where it is. We also remark the difference with respect to the more standard notion of promoters from [14]: promoters are multisets of objects that are used to activate a whole set of rules; the activated rules are then applied in a non-deterministic maximally parallel manner irrespectively of the multiplicity of the promoting multisets.

Let $\mathcal{T} = (V, E, w_1, \ldots, w_n, R, c_I)$ be a TPCU of degree $n \geq 1$. A configuration of $\mathcal{T}$ is any tuple $(w'_1, w'_2, \ldots, w'_n)$ with $w'i_i \in V^*$, for all $1 \leq i \leq n$. Then, given a multiset $x \in V^*$, the TPCU $\mathcal{T}$ *recognizes* multiset x if, by starting from configuration $(w_1, \ldots, x\, w_I, \ldots, w_n)$, after a finite sequence of transitions obtained by applying the rules as described above, it produces a final configuration where no more rules can be applied to the objects placed inside the cells and the environment. Moreover, the TPCU $\mathcal{T}$ recognizes a family of multisets M if, for

all $x \in M$, $\mathcal{T}$ recognizes x. If that is the case, we also say that the TPCU $\mathcal{T}$ recognizes the set of natural numbers $N(M) = \{ |x| \mid x \in M \}$.

Finally, we naturally associate to each TPCU a *communication graph* which represent the structure of the system as it is induced by the rules provided in the definition of the system.

Definition 2. *Let $\mathcal{T} = (V, E, C_1, \ldots, C_n, R, c_O)$ be a TPCU. The* communication graph of $\mathcal{T}$*, denoted by $\Gamma(\mathcal{T})$, is the undirected graph $(\{1, \ldots, n\}, A)$ where $\{i, j\} \in A$ if and only if, there is a rule $(i, a \to j, b)$ or $(i, b, a \to j)$ in R for some $a, b \in O$.*

Thus, given a TPCU, its communication graph is the graph containing a node per each cell in the system and an edge between every two cells which are allowed to interact by means of some rule. As we will see, this notion is particularly useful to graphically represent the structure of a given TPCU.

In the next section, we introduce some macro-elements (*macros*, for short) that group several conditional uniport rules. Macros are seen as sub-functional units which can be combined to form larger "blocks of cells" dedicated to perform some specific tasks. Specifically, we will define macros necessary to construct blocks which simulate incrementing and decrementing instructions of a deterministic register machine. Thus, in Section 5, we will be able to show the computational completeness of TPCU's.

4 Macros

Here we present the details of the macros mentioned at the end of the previous section; we also introduce a specific graphical representation for them. We remark that the behavior of each macro is non-deterministic, but instructions are grouped in such a way that macros have only one non-looping branch in the non-deterministic evolution. Then, after the description of these macros, we show how they can be combined to construct simulation blocks for the incrementing and decrementing instructions of a deterministic register machine.

4.1 Synchronization of Two Signals

This macro, shortly denoted as syn2, is represented in the left part of Figure 1.

This macro aims to synchronize symbols s_1 and s_2 which are treated like signals. If both of them are present in input cells (1 and 2), then they will continue to output cells (3 and 4). More precisely, if object s_1 is present in cell 1 and object s_2 is present in cell 2, then object s_1 will go to cell 3 and object s_2 will go to cell 4. Notice that the opposite is true as well: if one of the two signals is missing, then the other one does not move. No assumption about the time necessary to do this operation is made, i.e., it is not done in one time unit. We can only affirm that s_1 arrives in cell 3 before s_2 arrives in cell 4. We give below necessary rules that implement the syn2 macro.

1. $(2, s_2 \to 1, s_1)$
2. $(1, s_2, s_1 \to 3)$

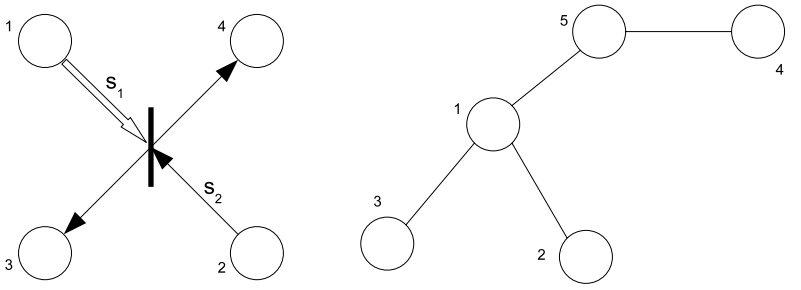


Fig. 1. syn2 element

3. $(5, X \to 1, s_2)$
4. $(5, X' \to 1, X)$
5. $(1, X' \to 5, s_2)$
6. $(1, X \to 5)$
7. $(1, X', s_2 \to 5)$
8. $(5, X', s_2 \to 4)$

Symbols X and X' are initially present in cell 5.

The communication graph induced by these rules is presented in the right part of Figure 1. It is easy to observe that the above structure, rules and initial objects permit to obtain the desired behavior. Indeed, if symbol s_1 is present in cell 1 and there is no symbol s_2 in cell 2, then nothing happens. Similarly, if s_2 is present and s_1 is not present, this part of the system does not evolve. When both s_1 and s_2 are present, rule 1 brings s_2 to cell 1. After that either rule 2, or rule 3 will be applied (but not both of them because s_2 cannot be involved in more than one rule). Suppose that rule 3 is applied (the case of rule 2 is similar). In this case, symbol X is brought to cell 1. At the next step, symbol s_1 is sent by rule 2 to cell 3, hence performing the first part of the synchronization. At the same time one of the rules 4 or 6 is applied. If rule 6 is applied, then the system may loop using rules 3 and 6. Hence, at some moment rule 4 will be applied. This application brings symbol X' in cell 1. This symbol permits to move s_2 to cell a and further to cell 4. At the same time symbols X and X' return to their original place in cell a and they are ready for another application of this macro-element.

We note that the synchronization of s_1 and s_2 is the only non-looping evolution of the subsystem above. Moreover, the same group of cells of Figure 1 with the same communication graph may be reused for other pairs of signals by just adding similar rules for each pair of signals to be synchronized.

4.2 Synchronization and Duplication of Two Signals

This macro, shortly denoted as syndup, is represented in the left part of Figure 2.

This macro, like the previous one, synchronizes symbols s_1 and s_2: if both of them are present in input cells (1 and 2), then they will continue to output cells (3 and 4). At the same time symbol s_3 goes from cell 5 to cell 6. If s_2 arrives

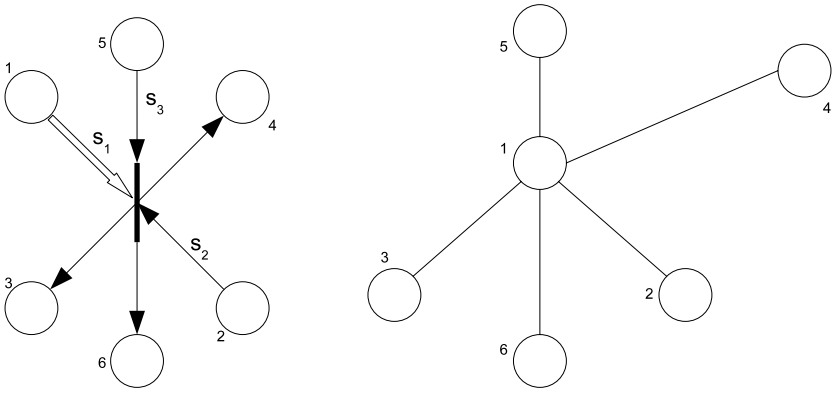


Fig. 2. syndup macro

to cell 4, it will be present there at least one step before s_3 arrives to cell 6. In some sense, s_3 is a copy of s_2 (which is a little bit time-shifted).

More precisely, if object s_1 (resp. s_2, s_3) is present in cell 1 (resp. cell 2, cell 5), then object s_1 will go to cell 3, object s_3 will go to cell 6 and object s_2 possibly will go to cell 4. If object s_2 is sent to cell 4, then it arrives there one step before s_3 arrives in cell 6. As before, no assumption about the time necessary to do this operation is made. We give below necessary rules that implement the syndup macro.

1. $(2, s_2 \to 1, s_1)$
2. $(1, s_2, s_1 \to 3)$
3. $(5, X \to 1, s_2)$
4. $(5, s_3 \to 1, X)$
5. $(1, X \to 5)$
6. $(1, s_3, s_2 \to 4)$
7. $(1, s_3 \to 6)$

Symbol X is initially present in cell 5.

The communication graph induced by these rules is presented in the right part of Figure 2.

Similarly to macro syn2, it is easy to observe that the above structure, rules and initial objects permit to obtain the desired behavior. Symbol s_1 is sent to cell 3, while symbol s_3 may send symbol s_2 to cell 4. Finally, symbol s_3 goes to cell 6. There are two non-looping evolutions of this macro-element corresponding to the final position of s_2.

We remark that cells 1, 2 and 3 may be shared between syn2 and syndup macros.

4.3 Infinite Loop

This macro, shortly denoted as i-loop, is represented in the left part of Figure 3.

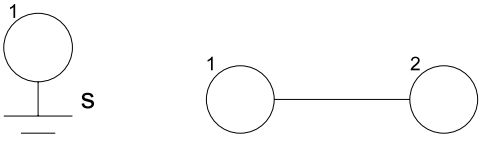


Fig. 3. i-loop macro

This macro has the following meaning. Object s is always a subject to a computation in cell 1. Hence, if it is not taken from there, the system will never halt. We give below the necessary rules that implement this macro.

1. $(1, s \to 2, X_s)$
2. $(2, X_s, s \to 1)$

Symbol X_s is initially present in cell 2.

It is clear that above rules will infinitely move symbol s between cells 1 and a.

4.4 Symbol Check

This macro-instruction, shortly denoted as s-check, is represented in the left part of Figure 4.

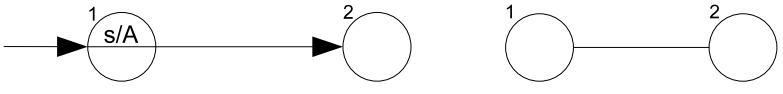


Fig. 4. s-check macro

This macro has the following meaning. In presence of object A in cell 1, symbol s from cell 1 will go to cell 2. This behavior is simply implemented by a rule $(1, A, s \to 2)$.

4.5 Incrementing and Decrementing a Counter

The left part of Figure 5 presents two macros denoted A-plus and A-minus.

These macros are used to increment/decrement the counter A. When object s (s_1) arrives to cell 1, it increments (decrements) counter A and after that s (s_1) goes to cell 2. We give below necessary rules that implement these macros (cell 0 is the environment):

1. $(3, A \to 1, s_1)$
2. $(0, A \to 1, s)$
3. $(1, s \to 6)$
4. $(1, s_1 \to 6)$
5. $(1, A \to 4)$
6. $(4, A \to 5)$
7. $(4, A \to 5, s)$

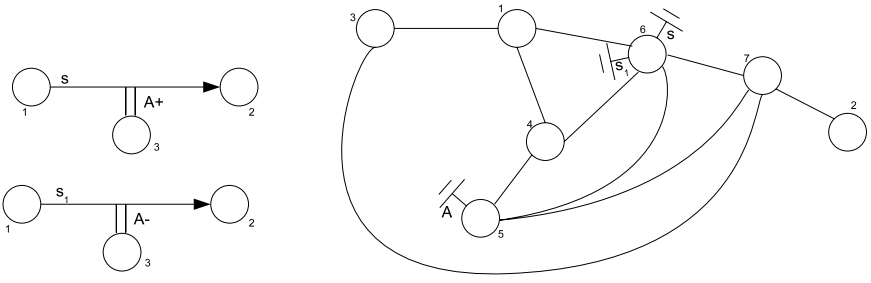


Fig. 5. A-plus and A-minus macros

8. $(4, A \to 5, s_1)$
9. $(6, A \to 5)$
10. $(6, A, s \to 7)$
11. $(6, A, s \to 7)$
12. $(6, A \to 7, s)$
13. $(6, A \to 7, s_1)$
14. $(7, A \to 5)$
15. $(7, s \to 2)$
16. $(7, s_1 \to 2)$
17. $(7, s, A \to 3)$
18. $(7, s_1, A \to 0)$

The idea used to implement these macros is quite simple: symbol s (resp. s_1) brings from the environment (resp. cell 3) symbol A into cell 1. If more than one A is brought, than at least one A will arrive in cell b triggering an infinite computation. Hence, the only possible halting computation is the one which brings an object A from the environment, at the next step moves A and s (s_1) from cell 1 to cell a and cell c respectively, and then uses s to move A from cell a to cell c. After that, in such a computation, both symbols are moved to cell d, and finally A will go to its place, while s (s_1) reaches cell 2.

Notice that, although A-plus and A-minus are seen as separate macros, they are actually implemented through a unique group of cells containing rules for simulating both an incrementing and a decrementing instruction. In general, the same group of cells with the communication graph of Figure 5 can be used to simulate many different incrementing/decrementing instructions by joining the sets of rules necessary to simulate each instruction.

Remark 1. Macros defined above may be joined just by superposing some of their cells. Hence, a net-like structure may be built using these elements. Moreover, since we place objects in cells and after that move them, the obtained system has similarities with Petri Nets (e.g., see [15]). However, there are significant differences. In particular, macros above are in some sense time-less, because there is no limit on the number of steps necessary to perform their action. But in a halting configuration this will finally happen.

5 Computational Completeness

In this section we show how a register machine is simulated by using the macros introduced in the previous section, and hence we obtain the computational completeness of TPCU's. To this aim, we introduce the notation $NRTPCU_n$, for any $n \geq 1$, to denote the family of sets of natural numbers recognized by tissue P systems with conditional uniport.

Theorem 1. $NRTPCU_{24} = NRE$.

Proof. Let M be a deterministic register machine. We will prove the assertion of the theorem in the following way. First, we construct Π and we show that this system simulates the behavior of M. In the same time, we investigate all other possible evolutions of Π and we show that only evolutions corresponding to the simulation of M lead to a halting configuration of Π.

In what follows we show how an arbitrary incrementing instruction $(p, A+, q)$ of M is simulated. We mentioned before that we use macros defined in Section 4 as building blocks. We combine them as it is depicted in Figure 6. We number cells and we indicate their number below them. We remark that rules and objects of Π can be easily deduced from these pictures.

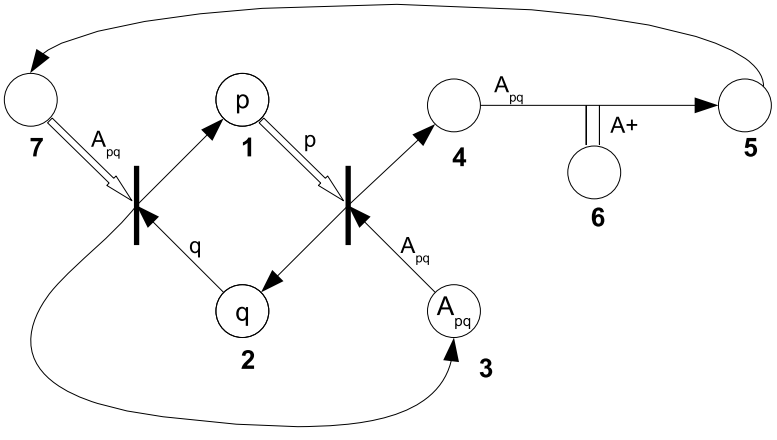


Fig. 6. Simulation of $(p, A+, q)$

The incrementing instruction is simulated as follows. Signals (symbols) p and A_{pq} synchronize, after that A_{pq} increments register A, and synchronizes with symbol q. After these actions q is exchanged with p, A_{pq} returns to its place and register A is incremented.

As we have already said, all incrementing instructions of M may be simulated using the same graph structure because macros for different instructions may share the same cells.

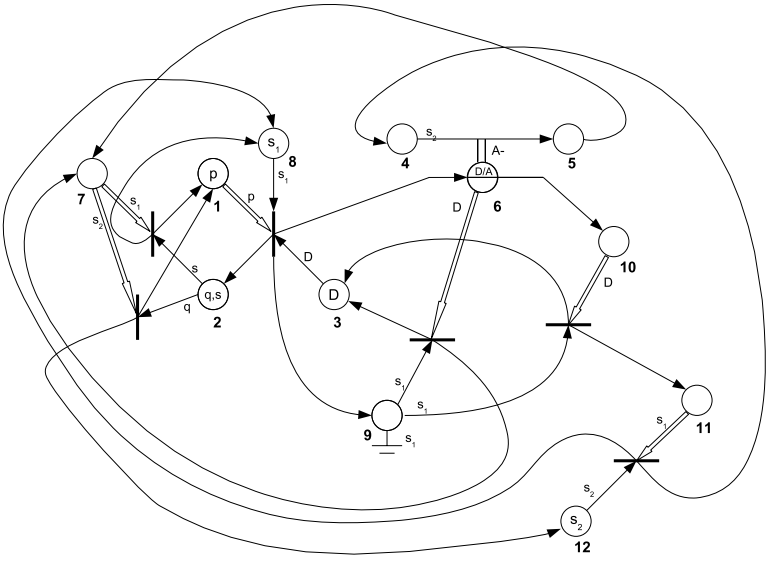


Fig. 7. Simulation of $(p, A-, q, s)$

The case of a decrementing instruction $(p, A-, q, s)$ is depicted in Figure 7 (we remark that all working symbols (D, s_1, s_2) will be indexed by the number of the instruction). We number cells and we indicate their number below them. Specifically, in order to simulate a decrementing instruction, signals p and D (D indeed stands for D_{pqs}) synchronize and if D does not arrive in the output cell 6 (we recall that it arrives there non-deterministically), then s_1 will be kept in an infinite computation (in cell 9). Hence this branch will not halt. In the other branch of the computation, D arrives in the output cell 6 and if there are symbols A present, then it will move further. We remark that cell 6 stores counter symbols. Now, the symbol s_1, depending on position of D will choose the corresponding branch of the computation. In this way the zero check on register A is performed. We note, that finally all additional symbols return to their original places. We remark that we use the same three cells (1, 2 and 7) for both signal synchronization in the upper left corner.

Like in the previous case the simulation of all decrementing instructions may share the same graph. Moreover, the incrementing and incrementing graphical representations have a common underlying graph (which is in fact the graph corresponding to the decrementing enriched with some edges). The initial configuration contains the symbol q_0 at cell 1.

We stress once more that Π can be easily deduced from the graphical representations corresponding to each instruction. From a structural point of view, cell 1 contains the current state, cell 2 contains all states of the machine except the current one. Cell 3 contains symbols A_{pq} and D_{pqs} that are used to drive

the all simulation in Π of corresponding instructions from M. Cell 6 contains unary values of all the registers. We also note that after expanding all macro-instructions, system Π contains 24 cells.

For any configuration of M $(q, A_1^{n_1}, \ldots, A_k^{n_k})$, there is a corresponding configuration of Π, where cell 1 contains q and cell 6 contains symbols $A_1^{n_1}, \ldots, A_k^{n_k}$. Then, in order to finish the proof we need to show that the simulation of M is the only possible halting computation. Indeed, we observe that the computation is started when a symbol p corresponding to a state of M arrives in cell 1 (initially in cell 1 symbol q_0 is present). Suppose, for simplicity, that there is the following instruction of M: $(p, A+, q)$. In this case, symbol p goes to cell 2 where all unused state symbols are kept. In the meanwhile symbol A_{pq} goes to cell 4 and drives the computation – first it increments register A and after that synchronizes with symbol q. At the end, symbol A_{pq} returns to cell 2, while cell 1 contains q, hence corresponding instruction of M is simulated.

For the decrementing case, symbol p is synchronized with D which drives the further computation. Therefore, at any moment, there is only one symbol that triggers the further computation (leading to a halting configuration). Since there is only one halting evolution, it corresponds to the simulation of M. □

6 Discussion

In this paper a class of tissue P systems, called "with conditional uniport" or TPCU's, for short, is introduced. This model relies on simple communication rules which move simple symbol objects between adjacent components either freely or in the presence of another symbol object in one of these components. It is proved that this model, although with these very simple rules, it is computationally complete: it can recognize all recursively enumerable sets of natural numbers. In this respect, we also conjecture that, with some modifications in the proof of Theorem 1, TPCU's can be shown to be able to simulate non-deterministic register machines. This could lead to a characterization of the family of recursively enumerable sets of natural numbers based on generative TPCU's which do not require an input multiset.

The idea of conditional communication is not completely new to the area of membrane computing. For instance, the concept of activated membrane channels was previously introduced in [6], and P automata [5] already considered rules which allow a multiset to enter a membrane only in presence of another specific multiset inside that membrane. However, in these variants, "activators" are defined as generic multisets of any size, and, in order to achieve the power of Turing machines, activators of size at least two are always used. Thus, our approach is closer to minimal symport/antiport [16] as it reports completeness for systems with a "minimal" cooperation in the communication rules: activators consists of one single object, and rules involve at most two objects. On the other hand, with respect to results reported in [16], the completeness of conditional uniport is obtained by using a higher number of cells and tissue P systems with

a complex graph structure. The optimality of this result is not known though, and this opens the possibility for improvements on the number of cells.

In addition to the completeness result the paper introduces a set of "blocks" of tissue P system components using conditional uniport rules with a certain behavior – the synchronization of the objects moving between components, incrementing/decrementing the number of object symbols when a specific 'signal' object occurs in the block. These constructions are useful in order to simplify the proof of Theorem 1, but they might be considered as primitive components that (maybe with some restrictions or modifications) are combined to produce more powerful blocks. In our future works this approach will be investigated as a modular way of building solutions to some problems. This proposal is very relevant for investigations into modeling self-assembly phenomena by using P systems and their variants [4], [7] as it shows a great suitability in this respect.

Acknowledgements

Marian Gheorghe and Francesco Bernardini are supported by the British Council research contract PN 05.014/2006, Maurice Margenstern and Sergey Verlan are funded by the Égide programme Alliance 0881UG. Francesco Bernardini is also supported by NWO, Organisation for Scientific Research of The Netherlands, project 635.100.006 "VIEWS".

References

1. Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., Walter, P.: *The Molecular Biology of the Cell.* 4th edn., Garland Publ. Inc., London, 2002.
2. Alhazov, A., Rogozhin, Y., Verlan, S.: Symport/Antiport Tissue P Systems with Minimal Cooperation. In *Proceedings of the ESF Exploratory Workshop on Cellular Computing (Complexity Aspects)*, Sevilla, Spain, 2005, 37–52.
3. Bernardini, F., Gheorghe, M.: Cell Communication in Tissue P Systems: Universality Results. *Soft Computing*, **9** (2005), 640–649.
4. Bernardini, F., Gheorghe, M., Krasnogor, N., Giavitto, J.L.: On Self-Assembly in Population P Systems. In Calude, C., Dinneen, M., Păun, G., Pérez-Jiménez, M.J., Rozenberg, G., eds.: *Uncoventional Computation. 4th International Conference, UC 2005*, Sevilla, Spain, October 2005, LNCS 3365, Springer, 2005, 46–57.
5. Csuhaj-Varjú, E., Vaszil, G.: P Automata or Purely Communicative Accepting P Systems. In Păun, G., Rozenberg, G., Salomaa, A., Zandron, C., eds.: *Membrane Computing. International Workshop, WMC-CdeA 02*, Curtea de Argeş, Romania, August 19-23, 2002, LNCS 2597, Springer, 2003, 219–233.
6. Freund, R., Oswald, M.: P Systems with Activated/Prohibited Membrane Channels. In Păun, G., Rozenberg, G., Salomaa, A., Zandron, C., eds.: *Membrane Computing. International Workshop, WMC-CdeA 02*, Curtea de Argeş, Romania, August 19-23, 2002, LNCS 2597, Springer, 2003, 261–269.
7. Gheorghe, M., Păun, G.: Computing by Self-Assembly: DNA Molecules, Polynominoes, Cells. In Krasnogor, N., Gustafson, S., Pelta, D., Verdegay, J.L., eds.: *Systems Self-Assembly: Multidisciplinary Snapshots. Studies in Multidisciplinarity*, Elsevier, 2006 (in press).

8. Goles, E., Margenstern, M.: Universality of the Chip-Firing Game. *Theoretical Computer Science*, **172** (1997), 91–120.
9. Margenstern, M.: Two Railway Circuits: a Universal Circuit and an NP-difficult one. *Computer Science Journal of Moldova*, **9** (2001), 3–33.
10. Martín-Vide, C., Păun, G., Rozenberg, G.: Membrane Systems with Carriers. *Theoretical Computer Science*, **270** (2002), 779–796.
11. Minsky, M.: *Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, New Jersey, 1967.
12. Păun, A., Păun, G.: The Power of Communication: P Systems with Symport/Antiport. *New Generation Computing*, **20** (2002), 295–305.
13. Păun, G.: Computing with Membranes. *Journal of Computer and System Sciences*, **61** (2000), 108–143.
14. Păun, G.: *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
15. Reisig, W.: *Petri Nets. An Introduction.* Springer, Berlin, 1985.
16. Rogozhin, Y., Alhazov, A., Freund, R.: Computational Power of Symport/Antiport: History, Advances and Open Problems. In Freund, R., Păun, G., Rozenberg, G., Salomaa, A., eds.: *Membrane Computing. International Workshop, WMC6*, Vienna, Austria, LNCS 3850, Springer, 2006, 1–31.
17. Rozenberg, G., Salomaa, A., eds.: *Handbook of Formal Languages.* Volume 1–3. Springer, 1997.
18. Toffoli, T., Margolus, N.: *Cellular Automata Machines.* The MIT Press, Cambridge, Mass., 1987.
19. The P Systems Web Page: `http://psystems.disco.unimib.it`

Distributed Evolutionary Algorithms Inspired by Membranes in Solving Continuous Optimization Problems

Daniela Zaharie[1] and Gabriel Ciobanu[2,3]

[1] Department of Computer Science, West University of Timişoara
Blvd. V. Pârvan no. 4, 300223 Timişoara, Romania
dzaharie@info.uvt.ro
[2] A.I. Cuza University, Faculty of Computer Science
Blvd. Carol I no.11, 700506 Iaşi
[3] Institute of Computer Science, Romanian Academy
Blvd. Carol I no. 8, 700505 Iaşi, Romania
gabriel@iit.tuiasi.ro

Abstract. In this paper we present an analysis of the similarities between distributed evolutionary algorithms and membrane systems. The correspondences between evolutionary operators and evolution rules and between communication topologies and policies in distributed evolutionary algorithms and membrane structures and communication rules in membrane systems are identified. As a result of this analysis we propose new strategies of applying the operators in evolutionary algorithms and new variants of distributed evolutionary algorithms. The behavior of these variants is numerically tested for some continuous optimization problems.

1 Introduction

Membrane systems and evolutionary algorithms are computation models inspired by nature, both based on applying some evolution(ary) rules to a (multi) set of simple or structured objects. Both models have distributed features. Membrane systems represent a suitable framework for distributed algorithms [2], and evolutionary algorithms allow natural extensions for distributed implementation [13].

A *membrane system* consists of a hierarchy of membranes that do not intersect, with a distinguishable membrane, called the *skin membrane*, surrounding them all. A membrane without any other membranes inside is *elementary*, while a non-elementary membrane is a *composite* membrane. The membranes produce a demarcation between *regions*. For each membrane there is a unique associated region. Because of this one-to-one correspondence we sometimes use membrane instead of region. 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. The multisets of objects from a region correspond to the "chemicals swimming in the solution in the cell compartment", while the rules correspond to the "chemical reactions possible in the same compartment". The

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 536–553, 2006.

rules must contain target indications, specifying the membrane where the new objects obtained after applying the rule are sent. The new objects either remain in the same region when they have a *here* target, or they pass through membranes, in two directions: they can be sent *out* of the membrane delimiting a region from outside, or can be sent *in* one of the membranes delimiting a region from inside, precisely identified by its label. In a step, the objects can pass only through one membrane. There exist many variants and classes of membrane systems; many of them are introduced in [8].

Evolutionary algorithms are reliable methods in solving hard problems in the field of discrete and continuous optimization. They are approximation algorithms which achieve a trade-off between solution quality and computational costs. Despite the large variety of evolutionary algorithms (genetic algorithms, evolution strategies, genetic programming, evolutionary programming), all of them are based on the same idea: evolve a *population* of candidate solutions by applying some rules inspired by biological evolution: *recombination (crossover)*, *mutation*, and *selection* [3]. An evolutionary algorithm acting on only one population is similar to a one-membrane system. *Distributed evolutionary algorithms*, which evolve separate but communicating (sub)populations, are more like membrane systems.

It is natural to ask questions as: *How similar are membrane computing and distributed evolutionary computing? Can ideas from membrane computing improve the evolutionary algorithms, or vice-versa?*

A first attempt to build a bridge between membrane computing and evolutionary computing is given by T.Y Nishida, in [7], where a membrane algorithm is developed by using a membrane structure together with ideas from genetic algorithms (crossover and mutation operators) and from metaheuristics for local search (tabu search).

In this paper we go further and deeper, and analyze the relationship between different membranes structures and different communication topologies specific to distributed evolutionary algorithms. Moreover, we propose new strategies for applying evolutionary rules and new variants of distributed evolutionary algorithms inspired by membrane systems structure and functioning.

The paper is organized as follows. In Section 2 we analyze the correspondence between evolutionary operators and evolution rules. As a result of this analysis, we propose a new, more flexible, strategy of applying the evolutionary operators. Section 3 is devoted to the similarities between membrane structures and communication topologies on one hand, and between communication rules in membrane systems and communication policies in distributed evolutionary algorithms on the other hand. As a result of this analysis is proposed a new variant of distributed evolutionary algorithms. Section 4 is devoted to a numerical analysis of the membrane systems inspired variants of evolutionary algorithms.

2 Evolutionary Operators and Evolution Rules

Evolutionary algorithms (EAs) working on only one population (panmictic EAs) can be interpreted as particular membrane systems having only one membrane.

Inside this single membrane there is a population of candidate solutions for the problem to be solved. Usually a population is an m-uple of n-dimensional vectors: $P = x_1 \dots x_m$, $x_i = (x_i^1, \dots, x_i^n) \in D$, where D is a discrete or a continuous domain depending on the problem to be solved. The evolutionary process consists in applying the recombination, mutation and selection operators to the current population in order to obtain a new population.

Recombination (crossover): The aim of this operator is to generate new elements from a set of elements (called parents) selected from the current population. Thus we have a mapping $\mathcal{R} : D^r \to D^q$, where usually $q \leq r$. Typical examples of recombination operators are:

$$r = q = 2, \qquad \mathcal{R}((u^1, \dots, u^n), (v^1, \dots, v^n)) = ((u^1, \dots, u^k, v^{k+1}, \dots, v^n), (v^1, \dots, v^k, u^{k+1}, \dots, u^n)) \tag{1}$$

and

$$r \text{ arbitrary }, q = 1, \qquad \mathcal{R}(x_{i_1}, \dots, x_{i_r}) = \frac{1}{r} \sum_{j=1}^{r} x_{i_j} \tag{2}$$

The first type of recombination corresponds to one point crossover (where $k \in \{1, \dots, n-1\}$ is an arbitrary cut point) used in genetic algorithms, while the second example corresponds to intermediate recombination used in evolution strategies [3].

Mutation: The aim of this operator is to generate a new element by perturbing one element from the current population. This can be modelled by a mapping $\mathcal{M} : D \to D$ defined by $\mathcal{M}((u^1, \dots, u^n)) = (v^1, \dots, v^n)$. Typical examples are:

$$v^i = \begin{cases} 1 - u^i & \text{with probability } p \\ u^i & \text{with probability } 1 - p \end{cases} \qquad \text{and} \qquad v^i = u^i + N(0, \sigma^i), \quad i = \overline{1, n} \tag{3}$$

The first example is used in genetic algorithms based on a binary coding ($u^i \in \{0, 1\}$), while the second one is typical for evolution strategies. $N(0, \sigma^i)$ denotes a random variable with normal distribution, of zero mean and standard deviation σ^i.

Selection: It is used to construct a set of elements starting from the current population such that the best elements with respect to the objective function of the optimization problem to be solved are favored. It does not generate new configurations, but only sets of existing (not necessarily distinct) configurations. Thus it maps D^q to D^r and can be used in two main situations: selection of parents for recombination (in this case $q = m$, $r < m$, and the parents selection is not necessarily based on the quality of elements), and selection of survivors (in this case $q \geq m$, $r = m$, and the survivors are stochastically or deterministically selected by taking into account their quality with respect to the optimization problem). In the following, the mapping corresponding to parents selection is

denoted by $\mathcal{S}_p$ and the mapping corresponding to survivors selection is denoted by $\mathcal{S}_s$.

A particular evolutionary algorithm is obtained by combining these evolutionary operators and by applying them iteratively to a population. Typical ways of combining the evolutionary operators lead to the main evolutionary strategies: generational, and steady state. In the generational (synchronous) strategy, at each step a population of new elements is generated by applying recombination and mutation. The population of the next generation is obtained by applying selection to the population of new elements or to the joined population of parents and offsprings. The general structure of a generational EA is presented in Algorithm 1 where $X(t)$ denotes the population corresponding to generation t, z denotes an offspring and Z denotes the population of offsprings. The symbol $\cup_+$ denotes an extended union, which allows multiple copies of the same element (as in multisets). The mapping $\overline{\mathcal{M}}$ is the extension of $\mathcal{M}$ to D^q, i.e., $\overline{\mathcal{M}}(x_{i_1}, \ldots, x_{i_q}) = (\mathcal{M}(x_{i_1}), \ldots, \mathcal{M}(x_{i_q}))$.

Algorithm 1. Generational Evolutionary Algorithm

1: Random initialization of population $X(0)$
2: $t := 0$
3: **repeat**
4: $\quad Z := \emptyset$
5: $\quad$ **for all** $i \in \{1, \ldots, m\}$ **do**
6: $\quad\quad Z := Z \cup_+ (\overline{\mathcal{M}} \circ \mathcal{R} \circ \mathcal{S}_p)(X(t))$
7: $\quad$ **end for**
8: $\quad X(t+1) := \mathcal{S}_s(X(t) \cup_+ Z)$
9: $\quad t := t + 1$
10: **until** a stopping condition is satisfied

In the steady state (asynchronous) strategy, at each step a new element is generated by recombination and mutation, and assimilated into the population if it is good enough (e.g., better than one of its parents, or than the worst element in the population). More details are in Algorithm 2.

The simplest way to interpret a generational or a steady state evolutionary algorithm as a membrane system is to consider the entire population as a structured object in a membrane, and the compound operator applied as one evolution rule which includes recombination, mutation and selection. Such an approach represents a rough and coarse handling which does not offer flexibility. A more flexible approach would be to consider each evolutionary operator as an evolution rule.

Evolutionary operators are usually applied in an ordered manner (as in Algorithms 1 and 2): first parents selection, then recombination and mutation, and finally survivors selection. Starting from the way the evolution rules are applied in a membrane system, we consider that *the rules can be independently applied to the population elements*, meaning that no predefined order between the operators is imposed. At each step any operator can be applied, up to some restrictions

Algorithm 2. Steady State Evolutionary Algorithm

1: Random initialization of population $X(0)$
2: $t := 0$
3: **repeat**
4: $\quad z := (\overline{\mathcal{M}} \circ \mathcal{R} \circ \mathcal{S}_p)(X(t))$
5: $\quad X(t+1) := \mathcal{S}_s(X(t) \cup_+ z)$
6: $\quad t := t + 1$
7: **until** a stopping condition is satisfied

ensuring the existence of the population. The recombination and mutation operators $\mathcal{R}$ and $\mathcal{M}$ can be of any type, with possible restrictions imposed by the coding of population elements. By applying these operators, new elements are created. These elements are unconditionally added to the population. Therefore by applying the recombination and mutation operators, the population size is increased. When the population size reaches an upper limit (e.g., twice the initial size of the population), then the operators $\mathcal{R}$ and $\mathcal{M}$ are inhibited.

The role of selection is to modify the distribution of elements in the population by eliminating or by cloning some elements. Simple selection operators could be defined by eliminating the worst element of the population, or by cloning the best element of the population. When selection is applied by cloning, then the population size is increased and selection is inhibited whenever the size reaches a given upper bound. On the other hand, when selection is applied by eliminating the worst element, the population size is reduced, and selection is inhibited whenever the size reaches a given lower bound (e.g., half of the initial size of the population).

By denoting with $x_1 \dots x_m$ ($x_i \in D$) the entire population, with $x_{i_1} \dots x_{i_q}$ an arbitrary part of the population, with x_* the best element and with x_- the worst element, the evolutionary operators can be described more in the spirit of evolution rules from membrane systems as follows:

Rule 1 (recombination): $x_{i_1} \dots x_{i_r} \to x_{i_1} \dots x_{i_r} x'_{i_1} \dots x'_{i_q}$ where $(x_{i_1}, \dots, x_{i_r}) = \mathcal{S}_p(x_1, \dots, x_m)$ is the set of parents defined by the selection operator $\mathcal{S}_p$, and $(x'_{i_1}, \dots, x'_{i_q}) = \mathcal{R}(x_{i_1}, \dots, x_{i_r})$ is the offspring set obtained by applying the recombination operator $\mathcal{R}$ to this set of parents;

Rule 2 (mutation): $x_i \to x_i x'_i$ where $x'_i = \mathcal{M}(x_i)$ is the perturbed element obtained by applying the mutation operator $\mathcal{M}$ to x_i;

Rule 3a (selection by deletion): $x_- \to \lambda$, meaning that the worst element (with respect to the objective function) is eliminated from the population;

Rule 3b (selection by cloning): $x_* \to x_* x_*$ meaning that the best element (with respect to the objective function) is duplicated.

Rule 4 (insertion of random elements): $x \to x\xi$, where $\xi \in D$ is a randomly generated element and x is an arbitrary element of the population.

The last rule does not correspond to the classical evolutionary operators but is used in evolutionary algorithms in order to stimulate the population diversity.

By following the spirit of membrane computing, these rules should be applied in a fully parallel manner. However, in order to avoid going too far from the classical way of applying the operators in evolutionary algorithms, we consider a sequential application of rules. Thus we obtain an intermediate strategy: the evolutionary operators are applied sequentially, but in an arbitrary order. Such a strategy, based on a probabilistic decision concerning the operator to be applied at each step, is described in Algorithm 3. The rules involved in the evolutionary process are: recombination, mutation, selection by deletion, selection by cloning and random elements insertion. The probabilities corresponding to these rules are p_R, p_M, p_{Sd}, p_{Sc} and $p_I \in [0, 1]$. By applying the evolutionary operators in such a probabilistic way, we obtain a flexible algorithm which works with variable size populations. In Algorithm 3 the population size corresponding to iteration t is denoted by $m(t)$. Even if variable, the population size is limited by a lower bound, m_*, and an upper bound, m^*.

Algorithm 3. Evolutionary algorithm with random selection of operators

Random initialization of the population $X(0) = x_1(0) \dots x_{m(0)}(0)$
$t := 0$
repeat
  generate a uniform random value $u \in (0, 1)$
  if $(u < p_R) \wedge (m(t) < m^*)$ **then**
    apply *Rule 1 (recombination)*
  end if
  if $(u \in [p_R, p_R + p_M)) \wedge (m(t) < m^*)$ **then**
    apply *Rule 2 (mutation)*
  end if
  if $(u \in [p_R + p_M, p_R + p_M + p_{Sd})) \wedge (m(t) > m_*)$ **then**
    apply *Rule 3a (selection by deletion)*
  end if
  if $(u \in [p_R + p_M + p_{Sd}, p_R + p_M + p_{Sd} + p_{Sc})) \wedge (m(t) < m^*)$ **then**
    apply *Rule 3b (selection by cloning)*
  end if
  if $(u \in [p_R + p_M + p_{Sd} + p_{Sc}, 1]) \wedge (m(t) < m^*)$ **then**
    apply *Rule 4 (insertion of a random element)*
  end if
  $t := t + 1$
until a stopping condition is satisfied

An important feature of Algorithm 3 is given by the fact that only one operator is applied at each step, and thus it can be considered as an operator oriented approach. This means that first an operator is probabilistically selected and only afterwards are selected the elements on which it is applied.

Another approach would be that oriented toward elements, meaning that at each step all elements can be involved in a transformation and for each one is selected (also probabilistically) the rule to be applied. After such a parallel step, a mechanism of regulating the population size can be triggered. If the population became too small, then selection by cloning can be applied or some

random elements could be inserted. If the population became too large, then selection by deletion could be applied. This strategy is characterized through a parallel application of rules, thus it is more in the spirit of membrane computing. However, this strategy did not provide better results than Algorithm 3 when it was tested for continuous optimization problems.

We can expect that the behavior of such algorithms be different from the behavior of more classical generational and steady state algorithms. However, from a theoretical viewpoint, such an algorithm can be still modeled by a Markov chain and the convergence results still hold [11]. This means that if we use a mutation operator based on a stochastic perturbation described by a distribution having a support which covers the domain D (e.g., normal distribution) and an elitist selection (the best element found during the search is not eliminated from the population), then the best element of the population converges in probability to the optimum.

The difference appears with respect to the finite time behavior of the algorithm, namely the ability to approximate (within a certain desired precision) the optimum in a finite number of steps. Preliminary tests suggest that for some optimization problems, the strategy with random selection of operators works better than the generational and steady state strategies; numerical results are presented in Section 4. This means that using ideas from the application of evolution rules in membrane systems, we can obtain new evolutionary strategies with different dynamics.

3 Communication Topologies and Policies

As it has been stated in the previous section, a one-population evolutionary algorithm can be mapped into a one-membrane system with rules associated to the evolutionary operators. Closer to membrane computing are the distributed evolutionary algorithms which work with multiple (sub)populations. In each subpopulation the same or different evolutionary operators can be applied leading to homogeneous or heterogeneous distributed EAs, respectively. Introducing a structure over the population has different motivations [13]: (i) it achieves a good balance between exploration and exploitation in the evolutionary process in order to prevent premature convergence (convergence to local optima) in the case of global optimization problems; (ii) it stimulates the population diversity in order to deal with multimodal optimization problems or with dynamic optimization problems; (iii) it is more suitable to parallel implementation.

Therefore, besides the possibility of improving the efficiency by parallel implementation, structuring the population in communicating subpopulations allows developing new search mechanisms which behave differently than their serial counterparts [13]. The *multi-population model of the evolutionary algorithms*, also called *island-model*, is based on the idea of dividing the population in some communicating subpopulations. In each subpopulation is applied an evolutionary algorithm for a given number of generations, then a migration process is started. During the migration process some elements can change their subpopulations,

or clones of some elements can replace elements belonging to other subpopulations. The main elements which influence the behavior of a multi-population evolutionary algorithm are the *communication topology* and the *communication policy*. The communication topology specifies which subpopulations are allowed to communicate while the communication policy describes how is ensured the communication. The communication topology in a distributed evolutionary algorithm plays a similar role as the membranes structure plays in a membrane system. On the other hand, the communication policy in distributed evolutionary algorithms is related to the communication rules in membrane systems.

3.1 Communication Topologies and Membrane Structures

The communication topology describes the connections between subpopulations. It can be modeled by a graph having nodes corresponding to subpopulations, and edges linking subpopulations which communicate in a direct manner. According to [1], typical examples of communication topologies are: fully connected topology (each subpopulation can communicate with any other subpopulation), linear or ring topology (only neighbor subpopulations can communicate), star topology (all subpopulations communicate through a kernel subpopulation). More specialized communication topologies are hierarchical topologies [5], and hypercube topologies [4]. The fully connected, star, and linear topology can be easily described by using hierarchical membrane structures which allows transferring elements either in a inner or in the outer membrane (see Figure 1).

Fully connected topology. Let us consider a number of s fully connected subpopulations. The fully connected topology can be modeled by using $s+1$ membranes, namely s elementary membranes and one skin membrane containing them (see Figure 1(a)). The elementary membranes correspond to the given s subpopulations, and they contain both evolution rules and communication rules. The skin membrane plays only the role of communication environment, thus it contains only communication rules and the objects which have been transferred from the inner membranes. The transfer of an element between two inner membranes is based on two steps: the transfer of the element from the source membrane to the skin membrane and the transfer of the element from the skin membrane to the target membrane. Another structure which corresponds to a fully connected topology is that associated to tissue P systems.

Star topology. The membrane structure corresponding to a star topology with s subpopulations is given by one skin membrane corresponding to the kernel subpopulation, and $s-1$ elementary membranes corresponding to the other subpopulations (see Figure 1(b)). The main difference from the previous structure associated to a fully connected topology is that the skin membrane has not only the role of an environment for communication, but it can contain also evolution rules.

Linear topology. In this case a subpopulation p can communicate only with its neighbor subpopulations $p+1$ and $p-1$. The corresponding structure is given

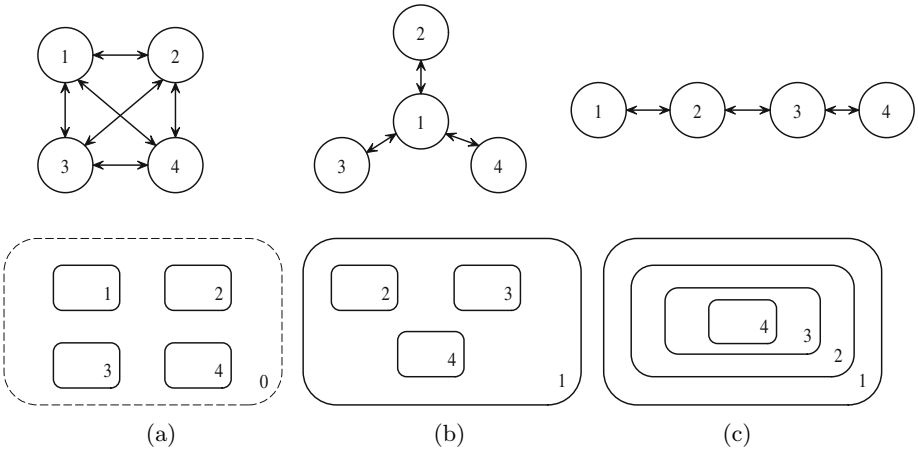


Fig. 1. Communication topologies in distributed evolutionary algorithms and their corresponding membranes structures. (a) Fully connected topology; (b) Star topology; (c) Linear topology.

by nested membranes, each membrane corresponding to a subpopulation (see Figure 1(c)).

Different situations appear in the case of ring and other topologies [4] which are associated with cyclic graph structures. In these situations the corresponding membrane structure is given by a net of membranes, or tissue P systems.

3.2 Communication Policies and Communication Rules

A communication policy refers to the way the communication is initiated, the way the migrants are selected, and the way the immigrants are incorporated into the target subpopulation. The communication can be initiated in a synchronous way after a given number of generations, or in an asynchronous way when a given event occurs. The classical variants of migrants selection are random selection and selection based on the fitness value (best elements migrate and the immigrants replace the worst elements of the target subpopulation). The communication policies are similar to communication rules in membrane computing, meaning that all communication steps can be described by some typical communication rules in membrane systems.

There are two main variants for transferring elements between subpopulations: (i) by sending a clone of an element from the source subpopulation to the target subpopulation (*pollination*); (ii) by moving an element from the source subpopulation to the target one (*plain migration*). An element is selected with a given probability, p_m, usually called migration probability. If the subpopulations size should be kept constant, then in the pollination case for each new incorporated element, another element (e.g., a random one, or the worst one)

is deleted. In the case of plain migration a replacing element (usually randomly selected) is sent from the target subpopulation to the source one.

In order to describe a random pollination process between s subpopulations by using communication rules specific to a membrane system, we consider the membrane structure described in Figure 1(a). Each elementary membrane corresponds to a subpopulation, and besides the objects corresponding to the elements in the subpopulation, it also contain some objects which are used for communication. These objects, denoted by r_{id}, are identifiers of the regions with which the subpopulation corresponding to the current region can communicate (in a fully connected topology of s subpopulations the identifiers belong to $\{1, \ldots, s\}$). On the other hand, when the migration step is initiated, a given number of copies of a migration symbol η are created into each elementary membrane. The multiplicity of η is related with the migration probability p_m (e.g., it is $\lfloor mp_m \rfloor$, where m is the size of subpopulation in the current region). Possible communication rules, for each type of membrane, describing the pollination process are presented in the following:

Elementary membranes. Let us consider the membrane corresponding to a subpopulation S_i ($i \neq 0$). There are two types of rules: an *exporting* rule ensuring the transfer of an element to the skin membrane which plays the role of an communication environment, and an *assimilation* rule ensuring, if it is necessary, that the subpopulation size is kept constant.
The export rule can be described as:

$$R^{S_i}_{\text{export}} : \eta x^{S_i} r^{S_i}_{id} \rightarrow (x^{S_i}, here)(x^{S_i} r^{S_i}_{id} d, out) \tag{4}$$

The assimilation rule can be described as:

$$R^{S_i}_{\text{ass}} : dx^{S_i} \rightarrow \lambda \tag{5}$$

x^{S_i} denotes in both rules an arbitrary element from the subpopulation S_i, and $r^{S_i}_{id}$ identifies the region where clones of the elements from the subpopulation S_i can be sent. At each application of $R^{S_i}_{\text{export}}$ a copy of the symbol η is consumed, and a copy of a deletion symbol d is created in the skin membrane.

Skin membrane. The communication rule corresponding to the skin membrane is:

$$R^0 : dx^{S_i} r^{S_i}_{id} \rightarrow (dx^{S_i}, in_{id}) \tag{6}$$

In the case of plain random migration, any element x^{S_i} from a source subpopulation S_i can be exchanged with an element x^{S_j} from a target subpopulation S_j. Such a communication process is similar with that in tissue P systems [8] described as $(i, x^{S_i}/x^{S_j}, j)$. Other communication policies (e.g., those based on elitist selection or replacement) can be similarly described.

3.3 Distributed Evolutionary Algorithms Inspired by Membrane Systems

A first communication strategy inspired by membrane systems is that used in the membrane algorithm proposed in [7] and also in [6]. The membrane algorithm

proposed in [7] can be interpreted as a hybrid distributed evolutionary algorithm based on a linear topology (Figure 1c) and a tabu search heuristic. The basic idea of communication between membranes is that of moving the best element in the inner membrane and the worst one in the outer membrane. The skin membrane receives random elements from the environment. The general structure of such an algorithm in presented in Algorithm 4.

Algorithm 4. Distributed evolutionary algorithm based on a linear topology

for all $i \in \{1, \ldots, s\}$ **do**
  Random initialization of the subpopulation S_i
end for
repeat
  for all $i \in \{1, \ldots, s\}$ **do**
    Apply an EA to S_i for τ steps
  end for
  Apply local search to the best element in S_1
  for all $i \in \{1, \ldots, s-1\}$ **do**
    send a clone of the best element from S_i to S_{i+1}
  end for
  for all $i \in \{2, \ldots, s\}$ **do**
    send a clone of the worst element from S_i to S_{i-1}
  end for
  Add a random element to S_1
  for all $i \in \{1, \ldots, s\}$ **do**
    Delete the two worst elements of S_i
  end for
until a stopping condition is satisfied

Another communication topology, corresponding to a simple membrane structure but not very common in distributed evolutionary computing, is that of star type (Figure 1b). In the following we propose a hybrid distributed evolutionary algorithm based on this topology. Let us consider a membrane structure consisting of a skin membrane containing $s-1$ elementary membranes. Each elementary membrane i contains a subpopulation S_i on which an evolutionary algorithm is applied. This evolutionary algorithm can be based on a random application of rules. The skin membrane contains also a subpopulation of elements, but different transformation rules are applied here (e.g., local search rules instead of evolutionary operators). The communication is only between S_1 (corresponding to skin membrane) and the other subpopulations. The algorithm consists of two stages: an evolutionary one and a communication one which are repeatedly applied until a stopping condition is satisfied. The general structure is described in Algorithm 5.

The *evolutionary stage* consists in applying an evolutionary algorithm on each of the subpopulations in inner membranes for τ iterations. The evolutionary stage is applied in parallel to all subpopulations. The subpopulations in inner membranes are initialized only at the beginning, thus the next evolutionary

stage starts from the current state of the population. In this stage the only transformation of the population in the skin membrane consists in applying a local search procedure to the best element of the population.

The *communication stage* consists in sending clones of the best element from the inner membranes to the skin membrane by applying the rule $x_* \rightarrow (x_*, here)(x_*, out)$ in each elementary membrane. Moreover, the worst elements from the inner membranes are replaced with randomly selected elements from the skin membrane. If the subpopulation S_1 should have more than s elements, then at each communication stage some randomly generated elements are added. The effect of such a communication strategy is that the worst elements in inner membranes are replaced with the best elements from other membranes or with randomly generated elements. In order to ensure the elitist character of the algorithm, the best element from the skin membrane is conserved at each step. It represents the approximation of the optimum we are looking for.

Algorithm 5. Distributed evolutionary algorithm based on a star topology

1: **for all** $i \in \{1, \ldots, s\}$ **do**
2: Random initialization of the subpopulation S_i
3: **end for**
4: **repeat**
5: **for all** $i \in \{2, \ldots, s\}$ **do**
6: Apply an EA to S_i for τ steps
7: **end for**
8: Apply local search to the best element in S_1
9: Reset subpopulation S_1 (all elements in S_1 excepting for the best one are deleted)
10: **for all** $i \in \{2, \ldots, s\}$ **do**
11: send a clone of the best element from S_i to S_1
12: **end for**
13: add random elements to S_1 (if its size should be larger than s)
14: **for all** $i \in \{2, \ldots, s\}$ **do**
15: Replace the worst element of S_i with a copy of a randomly selected element from S_1
16: **end for**
17: **until** a stopping condition is satisfied

4 Numerical Results

The aim of the experimental analysis was twofold: (i) to compare the behavior of the evolutionary algorithms with random application of operators (Algorithm 3) and of those based on generational and steady-state strategies (Algorithms 1 and 2); (ii) to compare the behavior of distributed evolutionary algorithms based on linear and star topologies (Algorithm 4 and Algorithm 5) with that of an algorithm based on a fully connected topology and random migration [15].

The particularities of the evolutionary algorithm applied in each subpopulation and the values of the control parameters used in the numerical experiments are presented in the following.

Evolutionary operators. The generation of offsprings is based on only one variation operator inspired from differential evolution algorithms [12]. It combines the recombination and mutation operators, so an offspring $z_i = \mathcal{R}(x_i, x_*, x_{r_1}, x_{r_2}, x_{r_3})$ is obtained by

$$z_i^j = \begin{cases} \gamma x_*^j + (1-\gamma)(x_{r_1}^j - x_*^j) + F \cdot (x_{r_2}^j - x_{r_3}^j)N(0,1), & \text{with probability } p \\ (1-\gamma)x_i^j + \gamma U(a_j, b_j), & \text{with probability} 1-p, \end{cases} \tag{7}$$

where r_1, r_2 and r_3 are random values from $\{1, \ldots, m\}$, x_* is the best element of the population, $F \in (0, 2]$, $p \in (0, 1]$, $\gamma \in [0, 1]$ and $U(a_j, b_j)$ is a random value, uniformly generated in domain of values for component j.

In the generational strategy an entire population of offsprings $z_1 \ldots z_m$ is constructed by applying the above rule. The survivors are selected by comparing the parent x_i with its offspring z_i and by choosing the best one. In the sequential strategy, at each step is generated one offspring which replaces, if it is better, the worst element of the population.

In the variant based on Algorithm 3 the following rules are probabilistically applied: the recombination operator given by Equation (7) is applied with probability p_R, the selection by deletion is applied with probability p_{Sd} and the insertion of random elements is applied with probability p_I. Since there are two variants of the recombination operator (for $\gamma = 0$ and for $\gamma = 1$) each one can be applied with a given probability: p_R^0 and p_R^1. These probabilities satisfy $p_R^0 + p_R^1 = p_R$.

Test functions. The algorithms have been applied to some classical test functions (see Table 1) used in empirical analysis of evolutionary algorithms. All these problems are of minimization type, the optimal solution being $x^* \in D$ and the optimal value being $f^* \in [-1, 1]$. x^* and f^* have been randomly chosen for each test problem. In all these tests the problem size was $n = 30$. The domains values of decision variables are $D = [-100, 100]^n$ for sphere function, $D = [-32, 32]^n$ for Ackley's function, $D = [-600, 600]^n$ for Griewank's function and $D = [-5.12, 5.12]^n$ for Rastrigin's function.

Parameters of the algorithms. The parameters controlling the evolutionary algorithm are chosen as follows: $m = 50$ (population size), $p = F = 0.5$ (the control parameters involved in the recombination rule given in Equation (7)), $\epsilon = 10^{-5}$ (accuracy of the optimum approximation).

We consider that the search process is successful whenever it finds a configuration, x_*, for which the objective function has a value which satisfies $|f^* - f(x_*)| < \epsilon$ by using less than 500000 objective functions evaluations. The ratio of successful runs from a set of independent runs (in our tests the number of independent runs of the same algorithm for different randomly initialized populations was 30) is a measure of the effectiveness of the algorithm. As a measure of efficiency we use the number nfe of objective function evaluations, both average value and standard deviation.

Table 1. Test functions

Name	Expression
Sphere	$f_1(x) = f^* + \sum_{i=1}^{n}(x_i - x_i^*)^2$
Ackley	$f_2(x) = f^* - 20\exp\left(-0.2\sqrt{\frac{\sum_{i=1}^{n}(x_i - x_i^*)^2}{n}}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi(x_i - x_i^*))\right) + 20 + e$
Griewank	$f_3(x) = f^* + \frac{1}{4000}\sum_{i=1}^{n}(x_i - x_i^*)^2 - \prod_{i=1}^{n}\cos((x_i - x_i^*)/\sqrt{i}) + 1$
Rastrigin	$f_4(x) = f^* + \sum_{i=1}^{n}((x_i - x_i^*)^2 - 10\cos(2\pi(x_i - x_i^*))) + 10$

Table 2. Comparison of evolutionary rules applying strategies in a panmictic EA

Test	Generational		Steady state		Algorithm 3	
fct	Success	$\langle nfe\rangle$	Success	$\langle nfe\rangle$	Success	$\langle nfe\rangle$
f_1	30/30	27308±396	30/30	25223±469	30/30	24758± 1328
f_2	30/30	37803±574	30/30	35223±611	29/30	27461± 2241
f_3	30/30	29198±1588	30/30	27010±1016	28/30	20017 ± 1640
f_4	19/30	335518±55107	18/30	296111±49316	29/30	193741± 113126

Results. Table 2 presents comparative results for generational, steady state and the strategy based on random selection of operators (Algorithm 3). For the first two strategies the evolutionary operator described by Equation (7) was applied for $\gamma = 0$. For $\gamma = 1$ the success ratio of generational and sequential variants is much smaller, therefore these results are not presented. The probabilities for applying the evolutionary operators in Algorithm 3 were $p_R = 0.5$ ($p_R^0 = 0.35$,$p_R^1 = 0.15$), $p_{Sd} = 0.5$, $p_I = 0$. The initial population size was $m(0) = 50$ and the lower and upper bounds were $m_* = m(0)/2$ and $m^* = 2m(0)$ respectively.

The results of Table 2 suggest that for the functions f_1, f_2, f_3 the Algorithm 3 does not prove to be superior to generational and steady state strategies. However a significant improvement can be observed for function f_4 which is a difficult problem for EA based on recombination as in Equation (7). However, by changing the probability p involved in the recombination operator (e.g., $p = 0.2$ instead of $p = 0.5$) a good behavior can be obtained also by generational and steady state strategies. On the other hand, by dynamically adjusting the probabilities of applying the evolutionary operators the behavior of Algorithm 3 can be improved. For instance, if one choose $p_R = 0$, $p_{Sd} = 0.1$ and $p_I = 0.9$ whenever the average variance of the population is lower than 10^{-8}, then in the case of Ackley function the success ratio is 30/30 and $\langle nfe\rangle$ is 19312 with a standard deviation of 13862.

The second set of experiments aimed to compare the communication strategies inspired by membranes (Algorithm 4 and Algorithm 5) with a communication strategy characterized by a fully connected topology and a random migration of elements [15]. In all experiments the number of subpopulations was $s = 5$, the initial size of each subpopulation was $m(0) = 10$ and the number of evolutionary steps between two communication stages was $\tau = 100$. In the case of random migration the probability of selecting an element for migration was $p_m = 0.1$.

Table 3. Behavior of distributed EAs based on a generational EA

Test fct.	Fully connected topology and random migration		Linear topology (Algorithm 4)		Star topology (Algorithm 5)	
	Success	$\langle nfe \rangle$	Success	$\langle nfe \rangle$	Success	$\langle nfe \rangle$
f_1	30/30	30500±1290	30/30	30678±897	30/30	34943± 4578
f_2	30/30	41000±2362	30/30	41349±1864	30/30	51230± 8485
f_3	20/30	32500±2449	30/30	33013±3355	26/30	41628 ± 8353
f_4	7/30	158357±66647	4/30	95538±10610	24/30	225280 ± 132636

The results in Table 3 show that the communication strategy based on the linear topology (Algorithm 4) behaves almost similarly to the strategy based on fully connected topology and random migration. On the other hand, the communication strategy based on the star topology (Algorithm 5) has a different behavior, characterized by a slower convergence. This behavior can be explained by the higher degree of randomness induced by inserting random elements in the skin membrane. However this behavior can be beneficial in the case of difficult problems (e.g., Rastrigin) by avoiding premature convergence situations. In the case of Rastrigin's function the success ratio of Algorithm 5 is significantly higher than in the case of the other two variants.

Table 4. Behavior of distributed EAs based on the random application of evolutionary operators

Test fct	Fully connected topology and random migration		Linear topology (Algorithm 4)		Star topology (Algorithm 5)	
	Success	$\langle nfe \rangle$	Success	$\langle nfe \rangle$	Success	$\langle nfe \rangle$
f_1	30/30	42033±4101	30/30	59840±11089	30/30	98280± 20564
f_2	30/30	117033±80778	30/30	156363±103488	29/30	266783± 96539
f_3	15/30	51065±14151	17/30	72901±38654	21/30	111227 ± 58491
f_4	30/30	94822±22487	30/30	107412±25363	30/30	111752 ± 19783

The results in Table 4 show that by using the Algorithm 3 in each subpopulation the convergence is significantly slower for Ackley and Griewank functions while it is significantly improved in the case of Rastrigin function, both with respect to other distributed variants and with the panmictic algorithms used in the experimental analysis.

These results suggest that structuring the population as in membrane systems, and applying the evolutionary operators in an unordered manner, we obtain evolutionary algorithms with a new dynamics. This new dynamics leads to significantly better results for certain evolutionary operators and test functions (see results in Table 4 for Rastrigin's function). However the hybrid approach is not superior to the classical generational variant combined with a random migration for the other test functions. Such a situation is not unusual in evolutionary computing, being accepted that no evolutionary algorithm is superior to all the others with respect to all problems [14].

Algorithm 5 is somewhat similar to the membrane algorithm proposed by Nishida in [7]. Both are hybrid approaches which combine evolutionary search with local search, and are based on a communication structure inspired by membrane systems. However there are some significant differences between these two approaches:

(i) they use different communication topologies: linear topology in the membrane algorithm of [7] vs. star topology in Algorithm 5; therefore they use different membrane structures (see Figure 2);
(ii) they address different classes of optimization problems: combinatorial optimization vs. continuous optimization;
(iii) they are based on different evolutionary rules (genetic crossover and mutation in [7] vs. differential evolution recombination here), and different local search procedures (tabu search in [7] vs. Nelder-Mead local search [10] in the current approach);
(iv) they are characterized by different granularity: micro-populations (e.g., two elements) but a medium number of membranes (e.g., 50) in [7] vs. medium sized subpopulations (e.g., 10) but a small number of membranes (e.g., 5);
(v) they are characterized by different communication frequencies: transfer of elements between membranes at each step in the membrane algorithm vs. transfer of elements only after τ evolutionary steps have been executed (e.g., $\tau = 100$).

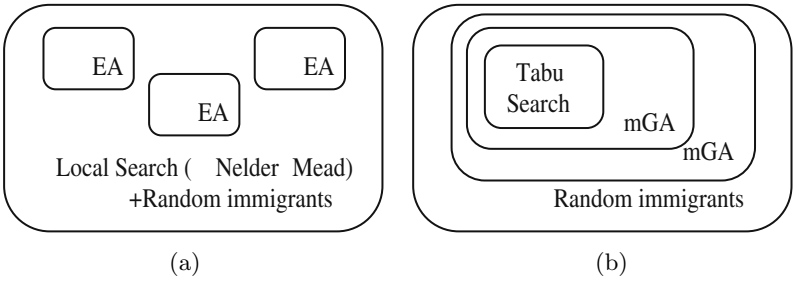


Fig. 2. (a) Membrane structure of Algorithm 5.(b) Membrane structure of Nishida's approach.

5 Conclusions

As it has been recently stated in [9], the membrane community is looking for a relationship, a link between membrane systems and distributed evolutionary algorithms. We claim that the main similarity is at a conceptual level, and each important concept in distributed evolutionary computing has a correspondent in membrane computing. This correspondence is summarized in the following table:

Membrane system	Distributed Evolutionary Algorithm
Membrane(region)	Population
Objects	Individuals
Evolution rules	Evolutionary operators
Membrane structure	Communication topology
Communication rules	Communication policy

Besides these conceptual similarities, there are some important differences:

(i) membrane systems have an exact notion of computation, while evolutionary computation is an approximate one;
(ii) membrane computing is based on symbolic representations, while evolutionary computing is mainly used together with numerical representations.

Despite these differences, ideas from membrane computing are useful in developing new distributed meta-heuristics. A first attempt was given by the membrane algorithm proposed in [7]. However this first approach did not emphasized at all the important similarities between membrane computing and distributed evolutionary computing. This aspect motivates us to start a depth analysis of these similarities, having the aim of describing the evolutionary algorithms by using the formalism of membrane computing. As a result of this analysis, we present in this paper a non-standard strategy of applying the evolutionary operators. This strategy, characterized by an arbitrary application of evolutionary operators, proved to be behave differently than the classical generational and steady state strategies when applied for some continuous optimization problems. On the other hand, based on the relationship between membrane structures and communication topologies, we introduce a new hybrid distributed evolutionary algorithm effective in solving some continuous optimization problems. The algorithms 3 and 5 proposed and analyzed in this paper are good and reliable in approximating solutions of optimization problems. This fact proves that by using ideas from membrane computing, new distributed metaheuristic methods can be developed.

Besides this way of combining membrane and evolutionary computing there are at least two other research directions which deserve further investigation:

(i) the use of evolutionary algorithms to evolve membrane structures;
(ii) the use of membrane systems formalism in order to understand the behavior of distributed evolutionary algorithms.

References

1. E. Alba, M. Tomassini. Parallelism and Evolutionary Algorithms, *IEEE Transactions on Evolutionary Computation*, **6**(5), pp. 443-462, 2002.
2. G. Ciobanu. Distributed Algorithms over Communicating Membrane Systems, *BioSystems* **70**(2), pp. 123–133, 2003.
3. A.E. Eiben, J.E. Smith. *Introduction to Evolutionary Computing*, Springer, 2002.
4. F. Herrera, M. Lozano. Gradual Distributed Real-Coded Genetic Algorithms, *IEEE Transactions on Evolutionary Computation*, **4**1, pp. 43–63, 2002.
5. J.J. Hu, E. D. Goodman. The Hierarchical Fair Competition (HFC) Model for Parallel Evolutionary Algorithms, *Proceedings of Congress of Evolutionary Computation*, IEEE Computer Society Press, pp. 49–54, 2002.
6. A. Leporati, D. Pagani. A Membrane Algorithm for the Min Storage Problem in H.J. Hoogeboom, Gh. Păun, G. Rozenberg (eds.), *Pre-Proceedings of the 7th Workshop on Membrane Computing*, 17-21 July 2006, Leiden, pp. 397–416, 2006.
7. T.Y. Nishida. An Application of P Systems: A New Algorithm for NP-complete Optimization Problems, in N. Callaos, et al. (eds.), *Proceedings of the 8th World Multi-Conference on Systems, Cybernetics and Informatics*, **V**, pp. 109–112, 2004.
8. Gh. Păun. *Membrane Computing. An Introduction*, Springer, 2002.
9. Gh. Păun. Further Twenty-Six Open Problems in Membrane Computing, *Third Brainstorming Meeting on Membrane Computing* (online document, `http://psystems.disco.unimib.it`), 2005.
10. W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling. *Numerical Recipes in C*, Cambridge University Press, 2002.
11. G. Rudolph. Convergence of Evolutionary Algorithms in General Search Spaces, in *Proc. of the third Congress on Evolutionary Computation*, IEEE Computer Society Press, pp. 50–54, 1996.
12. R. Storn, K. Price. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Technical Report TR-95-012*, ICSI, 1995.
13. M. Tomassini. Parallel and Distributed Evolutionary Algorithms: A Review, in K. Miettinen, M. Mäkelä, P. Neittaanmki and J. Periaux (eds.): *Evolutionary Algorithms in Engineering and Computer Science*, J. Wiley and Sons, pp. 113–133, 1999.
14. D.H. Wolpert, W.G. Macready. No Free Lunch Theorems for Optimization, *IEEE Transactions on Evolutionary Computing*, **1**, pp. 67–82, 1997.
15. D. Zaharie, D. Petcu. Parallel Implementation of Multi-population Differential Evolution, in D. Grigoras, A. Nicolau (eds.), *Concurrent Information Processing and Computing*, IOS Press, pp. 223–232, 2005.

Author Index