Gheorghe Păun
Grzegorz Rozenberg
Arto Salomaa
Claudio Zandron (Eds.)

LNCS 2597

# Membrane Computing

International Workshop, WMC-CdeA 2002
Curtea de Arges, Romania, August 2002
Revised Papers

Springer

Lecture Notes in Computer Science 2597

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
*Berlin*
*Heidelberg*
*New York*
*Barcelona*
*Hong Kong*
*London*
*Milan*
*Paris*
*Tokyo*

Gheorghe Păun Grzegorz Rozenberg
Arto Salomaa Claudio Zandron (Eds.)

# Membrane Computing

International Workshop, WMC-CdeA 2002
Curtea de Arges, Romania, August 19-23, 2002
Revised Papers

Springer

Volume Editors

Gheorghe Păun
Institute of Mathematics of the Romanian Academy
P.O. Box 1-764, 70700 Bucharest, Romania
Rovira i Virgili University, Research Group on Mathematical Linguistics
Pl. Imperial Tarraco, 1, 43005 Tarragona, Spain
E-mail: george.paun@imar.ro, gp@astor.urv.es

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

Arto Salomaa
Turku Centre for Computer Science
Lemminkäisenkatu 14A, 20520 Turku, Finland
E-mail: asalomaa@cs.utu.fi, asalomaa@utu.fi

Claudio Zandron
Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Via Bicocca degli Arcimboldi, 8, 20126 Milano, Italy
E-mail: zandron@disco.unimib.it

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

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

ISSN 0302-9743
ISBN 3-540-00611-7 Springer-Verlag Berlin Heidelberg New York



Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de


Printed in Germany

Typesetting: Camera-ready by author, data conversion by Steingräber Satztechnik GmbH, Heidelberg
Printed on acid-free paper SPIN: 10872548 06/3142 5 4 3 2 1 0

# Preface

This volume originated in the **Workshop on Membrane Computing, WMC-CdeA 2002**, which took place in Curtea de Argeş, Romania, during August 19–23, 2002. This was the third annual workshop held in Curtea de Argeş. The first one, **Workshop on Multiset Processing, WMP-CdeA 2000**, took place in August 2000, and the proceedings were published in Lecture Notes in Computer Science, volume 2235. The second one, **Workshop on Membrane Computing, WMC-CdeA 2001**, took place in August 2001, and selected papers were published as a special issue of *Fundamenta Informaticae*, volume 49, numbers 1–3, 2002.

The aim of these workshops is to provide a stimulating environment for researchers working in the area covered by a given workshop, so that existing scientific collaborations can be strengthened, and new collaborations (and friendships) can be initiated. Indeed, all three workshops held up to now were of such character, with very international attendance and collaboration taking place across national and scientific boundaries.

The 2002 Workshop, WMC-CdeA 2002, was the first workshop of the Molecular Computing Network (MolCoNet) funded by the EU Commission in the Fifth Framework program Information Society Technologies (project number IST–2001–32008). The preproceedings of WMC-CdeA 2002, Publication No. 1 of MolCoNet, were available at the meeting. The current volume differs considerably from the preproceedings: some of the papers from the preproceedings were not selected for this volume, while some papers were invited for this volume although they did not appear in the preproceedings. Moreover, all the papers from the preproceedings that were selected for this volume are significantly improved – the new versions reflect discussions that took place in Curtea de Argeş and the scientific collaborations that were initiated there (also, these papers went through an additional refereeing round).

Most of the papers are of a mathematical (theoretical computer science) nature, dealing with: the computational power (D. Besozzi et al.; F. Bernardini and V. Manca; M. Cavaliere; R. Freund and A. Păun; P. Frisco and H.J. Hoogeboom; M. Madhu and K. Krithivasan) and efficiency (E. Czeizler; M.J. Perez-Jimenez et al.) of membrane systems, applications (A. Atanasiu; G. Bel Enguix; G. Ciobanu et al.) and computer implementations/simulations (F. Arroyo et al.; D. Balbontin-Noval et al.), and links with other research areas (T. Bălănescu et al.). Some papers solve open problems from the literature (P. Sosík and R. Freund), or formulate new research topics (S. Marcus) or new approaches (R. Ceterchi and C. Martin-Vide; R. Freund and M. Oswald; P. Frisco and S. Ji; A. Obtulowicz). A number of papers provide mathematical (J.-L. Giavitto et al.; M. Kudlek and V. Mitrana) and biological backgrounds (I.I. Ardelean; R. Vasilco et al.). The original motivation for membrane systems came from the functioning of biological membranes, and although most of the current research

is oriented towards natural computing (more specifically, human-designed computing inspired by nature), it is hoped that in the long run the research on membrane computing will provide concepts and results useful for the understanding of the biology of membranes, and the role that membranes play in the functioning of living cells, and in communication between cells. Therefore, it is a nice alphabetical coincidence that the first and the last papers of the volume are authored by biologists, providing in this way the "biological bracketing" of the contents of this volume.

The fact that the WORKshop was really a place of interaction is witnessed by many papers with several co-authors. A convincing testimony to the creative atmosphere of the workshop is the number of papers co-authored by Rudi Freund. All these papers are the result of the intensive work that took place in Curtea de Argeş, in spite of the temptations for Rudi to spend more time with his daughter, Magdalena Franziska Patricia, 15 months in August 2002. In the "official diploma" that she received during the workshop dinner, she was qualified as "the most sensible result of the Workshops of Membrane Computing."

This MolCoNet workshop was organized by the Institute of Mathematics of the Romanian Academy, Bucharest, the University of Milano-Bicocca, Italy, and the "Vlaicu Vodă" National College of Curtea de Argeş, under the auspices of the European Molecular Computing Consortium. The workshop was also supported by the Institute of Microtechnology, Bucharest (grant PNCDI-MATNANTECH No. 68/2001 BIONANONET). The program committee consisted of Carlos Martin-Vide (Tarragona, Spain), Giancarlo Mauri (Milano, Italy), Gheorghe Păun (Bucharest, Romania, and Tarragona, Spain), Grzegorz Rozenberg (Leiden, The Netherlands), and Arto Salomaa (Turku, Finland).

The editing of this volume was supported by MolCoNet, and by the Rovira i Virgili University, Tarragona, Spain, where GP works as a researcher on the Ramon y Cajal program of the Spanish Ministry of Research.

November 2002

Gheorghe Păun
Grzegorz Rozenberg
Arto Salomaa
Claudio Zandron

# Table of Contents

# Molecular Biology of Bacteria and Its Relevance for P Systems

Ioan I. Ardelean

Institute of Biology of the Romanian Academy
Centre of Microbiology
Splaiul Independenţei 296
PO Box 56–53, Bucharest 79651, Romania
ioan.ardelean@ibiol.ro

**Abstract.** We recall several elements of molecular biology of bacteria, also discussing their (possible) relevance for the membrane computing area.

## 1 Introduction

Initiated by G. Păun in 1998, P systems are a branch of natural computing, rooted in the belief of its creator that "a formal computing device can be abstracted from the cell functioning" (Păun, 2001). Since then, the contributions to the development of P systems (see, for example, the web page http://psystems.disco.unimib.it or Păun, 2002) fully argue that the answer is positive. As the introduction of microbiology in P systems is still in its infancy (Ardelean, 2002) it would be interesting to elaborate on the relevance of molecular biology of bacteria on P systems. This is really needed because with the rapid progress in molecular biology (see the reviews by James, 1997; Boersema et al., 2001; Thieffry and Thomas, 1998; Walsburn and Yates, 2000; Raamsdonk et al., 2001; Devaux et al., 2001; Gunnevijk et al., 2001; Hess et al., 2001; Stormo and Tan, 2002; Phelps et al., 2002; Jong, 2002; Bruckner and Titgemeyer, 2002) and its marriage with computer science (reviews by Spengler, 2000; Meng et al., 2001; Kampfner, 2002; Marijuan, 2002; Ouzounis, 2002; Gaasterland and Oprea, 2002; Goodman, 2002; see also Conrad, 1972), new ideas come into the field of (micro)biology, some of them being directly related to P systems:

- Bray stresses that "many proteins in living cells appear to have as their primary function the transfer and processing of information, rather that the chemical transformation of metabolic intermediates or the building of cellular structures" (Bray, 1995).
- Hartwell et al. argue "for the recognition of functional modules as critical level of biological organization. Modules are composed of many types of molecules. They have discrete functions that arise from interactions among their components (proteins, DNA, RNA and small molecules) but these functions can not easily be predicted by studying the properties of the isolated components" (Hartwell et al., 1999).

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 1–18, 2003.

– Bruggeman et al., working on glutathione synthetase cascade in *E. coli*, showed "that this cascade is much more complex than necessary for simple regulation of ammonia assimilation. Simulations suggest that the function of this complexity may lie in quasi-intelligent behaviour, including conditioning and learning" (Bruggeman et al., 2000).

The aim of this paper is to discuss the relevance of molecular biology of bacteria to P systems, with special emphasis on the structure, hierarchy, and some functions of biological membranes, on the hierarchical control of functions in bacteria and on new questions that the microbiologists and P systems scientists will decide as to whether or not they deserve further attention.

The hope is that more P systems scientists will become more interested in biological realities and more microbiologists will start to learn about P systems. The work of both microbiologists who are acquainted with P systems and P systems scientists who have knowledge in microbiology will establish a real scientific bi-directional communication between the two scientific domains for the benefits of both of them. The reader interested in microbiological details is advised to consult classical books in microbiology as well as the references cited in this paper.

## 2 Cell Membrane Structure and Hierarchy in Bacteria

Bacteria are very small organisms the biological individual being composed of only one cell, so one say that they are unicellular (there are few interesting exceptions without relevance for the topic of this paper, however), as compared with more developed organisms (plants, animals and humans, for example) where the biological individual is composed of billions of cells.

In bacteria the cell is enclosed by a *cell wall* and a *cell membrane*, and contains cytoplasm and nucleoid. The cell membrane (CM) is basically composed of a lipid bilayer forming a semifluid matrix in which the membrane proteins are floating. This model of CM is called "fluid mosaic model" (Singer and Nicolson, 1972; Singer, 1974) and it is universally accepted. This is the basic structure of CM found in all biological cells. The huge diversity in CM belonging to different cells is related to the chemical composition of CM, namely the identity of proteins, carbohydrates, and lipids.

The general functions of CM, some of them having a strong relevance for P sytems, are the following:

1. CM serves as a selectively permeable barrier,
2. CM contains transport systems used for such tasks as nutrient uptake, waste secretion, and protein secretion,
3. CM holds the enzymatic machinery for crucial metabolic processes: respiration and photosynthesis,
4. CM synthesizes lipids and cell wall constituents,
5. CM contains special receptor molecules that help bacteria detect and respond to signal in their sourroundings thus affecting their behaviour.

I put forward that all the functions occurring at CM can have relevance for P systems; however, the P system scientists were so far more interested in transport, with special emphasis on symport/antiport (Păun et al, 2001; Păun and Păun, 2002; Martin-Vide et al., 2002; Frisco and Hoogeboom, 2002).

There are two main groups of bacteria according to the hierarchy of CM: *Gram-positive* and *Gram-negative* bacteria.

In Gram-positive bacteria the CM is found only at the exterior of the cytoplasm, separating the cell itself from the external medium.

In Gram-negative bacteria, apart from this CM found only at the exterior of the cytoplasm, there is a second CM, called *external membrane* (EM), because it is located at the exterior of first CM, a space being thus delimited in between CM and external membrane. This space is called periplasmic space and contains proteins involved in the transport of molecules from the external medium to the interior of the cell (see below) as well as other organic and inorganic components. The periplasmic space as well as intrathylakoidal space and cytoplasmatic space are examples of what in P systems are called regions. In the regions there are objects (P systems), actually, chemicals (microbiology), either organic (proteins, nucleic acids, lipids, carbohydrates) or inorganic (water, ions etc). The (spatial) arrangement of given chemicals/objects in the given space/region is one of the tasks to be done in the near future. It has to be stressed that, according to biological realities, as already noticed by Nishida (2002), within the membrane there are also objects, some of these objects (antiporters, symporters) being already in use in P systems, but the vast majority of them not yet considered in the P systems area.

The two membranes, EM and CM, with a structure described by the fluid mosaic model, have different chemical composition and different functions. For instance, there are some important differences in what concerns the transport of ions and molecules across them. Inogranic ions, for exemple, can pass through the EM while there are special proteins and mechanisms controlling their passage across CM (see below). Some molecules larger than inorganic ions, such us some antibiotics (an example is *valynomicin*), cannot cross the EM, thus offering to Gram-negative bacteria resistence to that antibiotic. Apart from the biological significance of the presence of EM in Gram-negative bacteria, for P systems the differences between CM and EM are an example for managing the communication through two membranes with the same structure but different chemical composition and functions.

Apart from CM and EM, in some bacteria there are some *intracellular membranes* (IM), organized in very tiny vesicles and associated with specific metabolic functions.

For exemple, in the cyanobacterium *Synechocystic* PCC 6803 (as well as in all other cyanobacteria with the exception of *Gloeobacter violaceus*), these vesicles are called thylakoides and are the site of the photosynthesis (see below). In *Rhodospirillum rubrum*, growing in light in the absence of molecular oxygen, the IM are also the site of photosynthesis. However, during the growth in the presence of molecular oxygen and in the absence of light, theses IM are no longer found,

thus the hierarchy of cell organization, is changed. In the presence of molecular oxygen, the cell gain energy by respiration at the CM, while in light in the absence of oxygen the main source of energy for the cell is photosynthesis. The ability of the same cell to grow either in light or in dark, is very important from a biological point of view; furthermore it could be relevant also for P systems as an example of changing the hierarchy of CM, a change possibly useful for considering some selfcontrolling P systems.

It has to be said that IM are not always connected to photosynthesis; in *Nitrosomonas*, for exemple, the IM are related to the oxydation of nitrite, a metabolic function by which this bacterium gains energy.

## 3 Processes Occurring at Biological Membranes

The processes occurring at biological membranes in bacteria (cell membrane, external membrane, or intracellular vesicles) are essential for cell life and they are also important not only to illustrate the impressive achievements in the molecular biology of bacteria during the last few decades, but also as natural examples for P system scientists of (types of) developmental rules which were not (fully) exploited yet in P systems. In this section, we will briefly point out some aspects related to the transport of chemicals across membranes as well as to respiration and photosynthesis.

### 3.1 The Transport of Ions and Molecules Across Cell Membrane

The transport is one fundamental function of CM; when the transport is drastically affected the composition of the cytoplasm changes dramatically and cell death shortly afterwards occurs.

The study of transport processes across CM in bacteria (as well as in other living cells) is under strong development, and the systems of classification as well. For the sake of simplicity, in accord with the aim of this paper, the main criteria for transport classification are presented (after Nicholls and Ferguson, 1992, simplified; for more information the reader is advised to consult Booth, 1988; Nicholls and Ferguson, 1992; Saier, 1999).

The transport can be *passive* or *active* (i.e., directly coupled to metabolism).

The transport is passive when either an ion or a molecule passes across the membrane from the compartment of a higher concentration to that with a lower concentration, the driving force of this movement being the concentration gradient of that ion or molecule, and there is no metabolical energy used for this transport. As an exemple of passive transport we mention the entry of sodium ions into a cell when, e.g., the cyanobacterium *Synechocystis* PCC 6803 cultivated in a normal medium containing around 50mM sodium ions is passed to a medium having 500mM sodium ions. Other substrates transported passively across CM are water, molecular oxygen, and carbon dioxide.

The transport is active when either an ion or a molecule passes across the membrane from the compartment of a lower concentration to that with a higher

concentration. There is the need for metabolical energy to accomplish this transport. Following the above exemple, in order to maintain a (relatively) constant composition of the cytoplasm with respect to sodium ion, the cell starts to transport sodium ions ouside it, from the compartment with a lower concentration (the cytoplasm) to the compartment with a higher concentration (the external medium) by consuming metabolic energy. Active transport is widely used by the cells because by this mechanisms the bacterial cells can maintain a relatively constant chemical composition of the cytoplasm, very different from that of the growing medium. It seems reasonable to consider that the active transport is more interesting for the development of P systems because the evolution rules are better illustrated by this type of transport than by passive transport. However, the significance of passive transport for P systems still waits to be confirmed.

When one deals with implementation of these modalities of transport in an abiotic structure (in silico?), passive and active mechanisms seem to be needed, as well. There is no living cell using only one type of transport, either active or passive, but in terms of P systems it could be of interest to check if a P system based totally either on passive transport or on active transport has any theoretical significance.

The transport can involve a charge transfer across the membrane (electrical transport) or not (electroneutral transport). Electrical transport is termed *electrogenic* ("creating a potential") when an electrically charged molecule or ion crosses the membrane; the simplest exemple of electrical transport is that of an ion (sodium, for example) across CM.

Electroneutral transport involves no net charge transfer across the membrane The simplest exemple of electroneutral transport is that of an uncharged molecule (glucose, for example) crossing the CM.

For electrical transport, it is calculated (Nicholls and Ferguson, 1992) that a single turnover of all electron-transport components in an individual bacterium will transport sufficient charge to establish a membrane potential approaching 200mV.

We believe that this aspect of transport is important for P systems, for instance, because if it is electrical energy to drive the future implemented P systems, as it seems reasonable to assume, then electrogenic versus electroneutral transport processes must be carefully handled.

A further interesting situation is the following: the transport of one ion or molecule across CM can be *coupled*. This means that it occurs only when another ion is transported across CM together with the first ion. There are two possibilities: i) both ions are transported in the same direction – this is called *symport*; ii) one ion is transported inside the cell and the other ion is simultaneously transported in the opposite direction, outside the cell – this is called *antiport*. Both these systems of transport are used by bacteria; the symport of protons with different substances nedeed for bacterial growth are very well documented (Jung, 2001). For example, *Escherichia coli* uses the symport of protons with either lactose, arabinose, or galactose. However, the proton is not the only type

of ions used with symport systems; sodium ions are also used for the symport of substances such as melobiose and proline.

In what concerns the antiport, a classical example is the proton/sodium antiporter found in many bacteria. Its major function is to maintain a rather constant concentration of either protons and sodium ions inside the cell (see Padan et al., 2001).

For P systems, symport and antiport are nice examples of how bacterial cells manage the developmental rules (Păun, 2000, 2001) and are already useful examples for the development of the theory (Păun et al., 2002; Păun and Păun, 2002; Martin-Vide et al., 2002; Frisco and Hoogeboom, 2002).

The P system s with symport/antiport received a special attention from P system scientists. In the about 12 months since such systems were first considred, several papers were published: Păun et al., 2001; Păun and Păun, 2002; Martin-Vide et al., 2002, Păun, 2002, Frisco and Hoogeboom, 2002; Freund, 2002, etc.

This topic is still attractive, at least for the following reasons:

- Symport and uniport, as well as other ways of transport found in different bacteria (Kelly and Thomas, 2001) could contribute to the further development of P systems theory.
- In the case of P systems based on symport/antiport only it is of interest to take more into account the biological realities of forbidding and permitting contexts. Learning from cells how they manage the function of each antiport system by using hierarchical control of cell functions, including the examples of forbidding and permitting contexts could help P system scientists to further refine their theories, for instance, with respect to the speed of calculations.
- The implementation of a P system can maybe make use of symport/antiport.

Strongly related to transport in bacteria there are the concepts of *delta* and *tau* actions used in P systems. *Delta* corresponds to the increase of passage of ions or molecules across the membrane, whereas *tau* corresponds to a decrease. To illustrate these concepts with microbial realities we will shortly discuss the mechanosensitive channels, a system (named *kdp*) for the transport of potassium ions and of sodium-proton antiporter.

Bacteria are able to maintain a relatively constant turgor pressure (the pressure exerted against the CM and cell wall) of about 15–20 atm in Gram-positive and 0.9–5 atm in Gram-negative bacteria. In *Escherichia coli*, for example, there are channels, named mechanosensitive channels, because they are sensitive against mechanical factors; they open during an osmotic downshock (the growing medium become suddently very diluted) (Levina et al., 1999) thus avoiding the excessive increase in the turgor pressure. This opening is reversible, and once the downshock gone, the channels close again.

One of the transport systems for potassium ions inside *Escherichia coli* (named the *kdp system*) is rapidly and irreversibly inhibited by moderate external potassium concentrations (Roe et al., 2000), thus preventing the overloading of cytoplasm with potassium ion.

$Na^+/H^+$ antiporter is also involved in maintaining in *E. coli* the pH of the cytoplasme around a value of about 7.5. When the external pH shifts from 7.2 to 8.3, the intracellular pH shortly increases to 8.3 and then the antiporter opens and increases the passage of both ions (*delta*): the protons come inside the cell and the sodium ions are extruded. The movement of both type of ions contributes to the acidification of the cytoplasm that, indeed, reach its normal pH of about 7.5 (for more details about the hierarchical control, see below).

These examples – and (micro) biology is very rich in other examples – not only prove the biological meaningfulness of *delta* and *tau* concepts from P systems, but also open the opportunity to use the theory of P systems as an alternative tool to study these biological events.

### 3.2 Respiration

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 molecular oxygen. The ability of many bacteria to use molecular oxygen as final electron acceptor in their respiration is provided by the work of an enzyme named *citocrom c oxidase*, which catalyzes the following equation:

$$O_{2\,in} + 4H^+_{in} + 4H^+_{in} + 4e^-_{within} = 2H_2O + 4H^+_{out}.$$

The subscript"in" means "on the inner face of the membrane","out" indicates the outer face of the membrane, while "within" simply means "within membrane".

Thus, during the last step of respiration shortly presented above water is formed from molecular oxygen, protons ($4H^+$) and electrons($4\,e^-$), and 4 protons are simultaneously transferred across membrane from inside to outside the cell contributing to energy conservation. Apart from its biological significance, the function of *citocrom c oxidase* could offer to P system scientists an example of a new type of developmental rule, more complex that those already considered in P systems. In a general formulation, this rule is:

$$A_{in} + B_{in} + C_{within} = D_{in} + B_{out}.$$

Moreover, the coefficients before the symbols could be of help in establishing of whether or not the function of *citocrom c oxidase* is an example of a biological computation.

Furthermore, the process of respiration involves a few other steps before that catalysed by *citocrom c oxidase*, each of them being an example of a new type of developmental rule not yet considered in P systems (Ardelean, manuscript in preparation).

### 3.3 Photosynthesis

The overall process of photosynthesis as it occurs in cyanobacteria (as well as in algae and plants) consists in using electrons from water to ultimately reduce

carbon dioxide thus forming substances such as carbohydrates. This process is essential for the life on Earth, being the main energy source for almost all living cells, including humans, the only source of molecular oxygen needed for respiration (and many oxygen-consuming related activities) as well as a huge carbon dioxide-consuming process.

The first major event in photosynthesis is splitting of water (at the expense of light energy, not presented for the sake of simplicity) to molecular oxygen, protons and electrons, according to the following equation:

$$2H_2O_{in} = 2H^+_{in} + O_{2in} + 4e^-_{within}.$$

Apart from its biological significance, the splitting of water could offer to P system scientists an example of a new type of developmental rule more complex that those already taken into account. In a general formulation, this rule is:

$$A_{in} = B_{in} + C_{in} + D_{within}.$$

Moreover, the process of photosynthesis includes a few other steps after the spitting of water, some of them being examples of a new type of P systems developmental rules (Ardelean, manuscript in preparation).

It is our belief that a more detailed presentation of respiration and photosynthsis, as well as of other biological processes not yet considered in P systems, could help to further develop the membrane computing by paying more attention to coefficients and energies related to processes such as those presented above; the same seems to be true with the examples from (micro)biology of what in P systems are called forbidding and permitting contexts (see below).

Considering the energetics of biological processes is very common in (micro)biology and it already started to receive some attention in the P systems area (Păun et al., 2001; Freund, 2002; Frisco and Ji, 2002; Alford, 2002). For P systems it would be interesting to pay more attention to the energetics of biological processes as this could be both of theoretical interest and can also provide hints on possible implementations of P systems. Related to energey there are the coefficients usually found with biochemical ecuations; perhaps such coefficients could be helpful in demonstrating that living cells really compute. Thus, we can move the answer to this question from personal opinions/intuitions, to an objective demonstration, changing our language from metaphoric to scientific, and moving from analogy – whatever important it is in stimulating creativity – to objectiv proofs.

Furthermore, related to energy, there is the reality of forbiding and permitting contexts, fluorishing in P systems. A wealth of examples of promoters/inhibitors can be found in (micro)biology, thus giving more support to the biological background and soundness of P systems.

Considering only the splitting of water, the presence of forbidding and permitting contexts are very illustrative. First, the splitting of water does not occur in darkness – there is an absolute need of light; even in light there is the need of other permitting conditions, such as:

- The presence of very complex biological machinery of photosynthesis, because the overall process of photosynthesis depends on the appropriate functions of different molecules organised in macromolecular structures within thylakoids.
- The availability of water in appropriate quantitites and concentrations (water solutions with high osmotic pressure allways inhibit photosynthesis).
- The availability of carbon dioxide to be reduced by the electrons and protons liberated during the splitting of water.

These permitting contexts allow both coarse and fine modulations of the overall process of photosynthesis, and learing from nature how to modulate functions, equations, and developmental rules would probably enhance the impact of P systems not only in the field of natural computing, but also in biology and in science in general.

There are also forbidding contexts such as the absence of specific or non specific inhibitors of water splitting or photosynthesis as a whole.

The interplay between permitting and forbidding contexts in tunning the activity of photosynthesis is a major trend in modern studies in photosynthesis and could add new insights to P systems theory. These studies could help P systems scientists to further refine the pioneering work on using P systems for simulation of photosynthesis (Nishida, 2002).

## 4 Hierarchical Control in Bacteria

The control of biochemical reactions taking place in a bacterial cell is performed at the following hierarchical levels: transcription, translation, and posttranslational modification. Transcription is the synthesis of messenger ribonucleic acid (mRNA) from a template of DNA, whereas translation is the synthesis of protein from an mRNA template. Posttranslational modification refers to covalent alterations of proteins after their translation from mRNA.

### 4.1 Transcriptional Control

In bacteria the majority of the genes are found grouped together in *operons*. The operon is a cluster of genes that encode proteins involved in the same metabolic process. One of the best known operon – and the first whose structure has been elucidated forty years ago by Jaques Monod, Francois Jacob and Andre Lwoff – is the *lac* operon. The lac operon contains three structural genes, each gene coding for one type of enzyme involved in the capability of the bacterium *E. coli* to use lactose as food: the *lacZ, lacY*, and *lacA* code for beta-galactosidase, galactosidase permease, and a transacetylase, respectively. Apart from the structural genes, the operon contains a control region that contains the promoter region and the operator region. When the operator region is blocked by a protein named *repressor* (coded by the repressor gene I situated outside the operon and NOT belonging to the operon itself), the structural genes are inactive and their corresponding proteins are not synthesised by the bacterium. This is the case where

in the growing medium there is glucose (alone or together with lactose, for example). After glucose depletion, the growth and multiplication of the cells ceased if no other carbon source is present; when lactose is present (in the absence of glucose), the repressor is no more acting as a repressor thus the information within structural genes is used for the synthesis of the corresponding proteins. Thus, in the presence of lactose, the cells become able to utilise this carbon source (for more details, see Stulke and Hillen, 1999; Bruckner and Titgemeyer, 2002).

The operon model, a fundamental development in the evolution of microbiology and molecular biology, could have major relevance for P systems. The arguments are as follows:

1. This model offers an example of forbidding and permitting contexts that are widely discussed in P systems.
2. This model offers an example of hierarchical control in bacteria together with the suggestion for the professionals of P systems to decide whether or not such hierarchical control would be of any use in P systems, for example, in improving the power of a P system, the efficiency, or the elegance of the calculus.
3. This model could argue that our knowledge on DNA, and in general on molecular biology, could be useful not only for DNA computing but also for P systems. More directly, we suggest that the very rich and rapidly advancing field of molecular biology can "feed" not only H systems but P systems, as well. Again, the P systems scientists must decide if this statement deserves their attention.

Nowadays it is known that the control of transcription occurs by very diverse mechanisms, acting at the level of either single operon or networks of operons.

Here are some details about the transcriptional regulation of individual operons:

1. Control of transcriptional *initiation* can be done by one of the following mechanisms:
   (a) The *promoter strength*, indicating its ability to capture RNA polymerase molecules and initiate transcription. This ability is significantly enhanced by activators whose activity changes when small molecular weight ligands bind to them. One of the best known activators is the cyclic AMP binding protein that plays a role in positive regulation of the *lac* operon.
   (b) The intervention of *alternate sigma factors* whose intracellular concentration changes during major events, such as stresses (osmotic, ionic etc.)
   (c) The *chemical modification of DNA* in the promoter region has, in general, a negative effect on transcription.
   (d) The *degree of supercoiling of DNA* affects its activity.
   (e) A *repressor protein* (encoded by a repressor gene, not belonging to the operon itself) inactivates the transcription (see *lac* operon).
   (f) *Autogenous regulation* concerns the regulation of genes that encode repressor protein that is sometimes brought about by their own gene product, the repressor they produce; the autogenous regulation thus provides a mechanism by which a repressor can prevent its own overproduction.

2. Control of transcription *termination* occurs when, for example, a specific protein (e.g., the factor Rho) blocks the activity of the RNA polymerase, thus receiving the name of transcription termination factor.

The presentation of these processes by which the process of transcription is controlled in bacteria aims to introduce the P systems scientists to the complexity of regulatory processes occurring in living cells at the level of each cell and opens the possibility to speculate about their significance for P systems.

The above processes show how the synthesis of mRNA is controlled both by coarse mechanisms (through a repressor protein, see the *lac* operon, as the classical example) and by fine mechanisms that modulates the synthesis of mRNA within a narrow value thus adjusting the concentration of mRNA to the needs of the cell at that moment.

Some of the above processes are illustrative of what in P systems is called forbidding and permitting conditions; furthermore, the fine mechanisms by which mRNA synthesis is controlled could suggest the introduction of fine tuning with respect to forbidding and permitting contexts also in P systems.

Introducing coarse (100% inhibition or 100% activity) forbidding and permitting contexts in P systems is very fruitful for P systems. What about fine forbidding and permitting contexts? How can they be captured in the P systems formalism? Would they be of any use? Intuitively, the implementation of the theory and the construction of a "P computer" may benefit from the possibility to control the rules both at coarse and fine levels.

Furthermore, mainly in the last decade, it became more evident that the response of a given bacterium to a change involves not only the activity of only one operon, but also the activities of many operons organised in networks (see van Bogelen et al., 1999; Thieffry and Thomas, 1998; Stormo and Tau, 2002; Phelps et al., 2002; Jong, 2002).

These networks involve hierarchical regulatory systems such as *regulons* and *modulons*. A regulon contains a few operons that are regulated by the same specific repressor or activator, whereas a modulon comprises several regulons and operons that are modulated by a common regulator; the control of this common regulator is superimposed on the control at the level of each individual operon or regulon. The common regulator responds to general conditions like nutrient starvation (carbon, nitrogen, phosphate, etc.) and other stress conditions (cold or heat shock, osmotic shift, oxidative stress) that demand major changes in the metabolism.

The global regulators enable bacteria to respond in a rapid and co-ordinated way to threats or opportunities presented by their environment (e.g., heat, cold, presence or absence of essential nutrients, high or low pH) by reconfiguring their biochemical machinery (McAdams and Arkin, 1998; Nougueira and Springer, 2000). The similarities between the logic of these genetic regulatory circuits and electronic digital logic in computer chips (McAdams and Arkin, 1998) could open the question about the capability of P systems theory to further improve the understanding of these hierarchic regulatory networks by using its own concepts, theorems etc.

The control of transcription by global regulators, apart of being a popular topic in contemporary microbiology could be instructive for P systems at least in the following directions:

1. The suggestion to control the developmental rules of a P system not only at the level of each individual/single rule but also at the level of sets or groups of rules eventually related to a larger mathematical operation within the process of calculation.
2. The suggestion to improve the response of a P system affected by a given challenge by mimicking the way a living cell "warms up" different parts of its capabilities when affected by an environmental challenge.

The above mechanisms are operating at different hierarchical levels within a cell, but bacteria have developed abilities to control transcription by mechanisms involving intercelullar communication through the excretion of defined chemicals.

These substances (N acyl-homoserine lactone and Peptide-pheromones in case of Gram-negative and Gram-positive bacteria, respectively) accumulate in the external medium and, when the concentration exceeds a threshold value, they change the expression of some genes, by a mechanism known as *quorum sensing* (Bassler, 1999). The quorum sensing participates in the self-regulation of cell densities (McAdams and Arkin, 1998).

For P systems, self-regulation of cell density discovered in biology is an example of how the function of a cell (P system) can be modulated by signals (peptides in the case with bacteria) produced by other (similar) cells (P systems); these molecules change the behaviour of other cells and are real examples of molecular switches. This example can lead to extensions of P systems, able to control the developmental rules within a network of P systems (cells) (Ardelean, 2002). At the same time, the quorum sensing, an ongrowing topic in microbiology (Bassler, 1999), argues for the importance of intercellular communication for P systems.

Quorum sensing also plays a critical role in the formation of bacterial biofilms where bacteria exhibit collective (multicellular) behaviour (Davies et al., 1998) beneficial with respect to protection from the environment, nutrient availability and metabolic co-operativity as well as acquisition of new genetic traits. In biofilms, bacteria localised at different regions display specialised patterns of gene expression and complex functional differentiation (Bassler, 1999).

This functional differentiation between cells could be suggestive of the functional differentiation of individual P systems organised in a larger structure (P network?) and/or for the operation in these differentiated P systems of different evolution rules. It also suggests the possibility of self-assembly of individual P systems, with or without differentiation within a larger structure.

### 4.2 Translational Regulation

The translational regulation of individual operon occurs mainly as a control of RNA stability and of translation initiation. This type of regulation is illustrated

for flagellum (Aldridge and Huges, 2002). The control of cell processes at this level could be interesting for P systems as an example of how a chemical object can be "in the range of" a rule A when it remains as it is, or must obey a rule B, when undergoes a minor change in its structure; these processes can occur either in the same compartment or in two compartments separated by a membrane.

### 4.3 Posttranslational Control and Modification of Proteins

The control of enzymatic activity within enzymatic pathways and the given pool of other metabolites is a very fast process (at the level of seconds), enabling the cells to adjust their metabolic activity to unexpected changes in the growth medium. The following are the main ways of control at this level:

1. *Reversible binding* to an enzyme of a small size molecule called *allosteric effector*, that changes the conformational state of the enzyme. It is assumed that the allosteric enzymes exist in two conformational states, one in which the active site of the enzyme has a high affinity for its substrate and another state which the active site of the enzyme has a low affinity for its substrate. Positive allosteric effectors increase enzymatic activity (by promoting the high affinity conformation) while negative allosteric effectors decrease enzymatic activities (by promoting the low affinity conformation). It seems that the property of allosteric enzymes to be either in a high activity conformation (when the positive allosteric effector is bound to the enzyme) or in a low activity conformation (when the negative allosteric effector is bound to the enzyme) can be an useful model for P systems: The object $a$ follows the rule A (active) when another object $b$ (positive allosteric effector) binds to it, but the object $a$ follows rule I (inactive) only when another object $c$ (negative allosteric effector) binds to it.
2. *Covalent modification* of an enzyme controls the enzyme activity by adding to or removing from the protein certain chemicals groups such as a phosphate. Is the reversible change of only one property of a given chemical object within a P system useful for membrane computing?
3. Control of biosynthetic pathways by *feedback inhibition*, which means that the end product of one metabolic pathway inhibits the first enzyme of that pathway. Again the question is what will happen to a computation when the first step of computation is affected, negatively or positively, by the last step?
4. Control by the *occurrence of isozymes* that are proteins that catalyse exactly the same chemical reaction.
5. The *energy* available within a cell greatly affects the enzymatic activities; briefly, high levels of energy inhibit the energy producing reactions and stimulate the energy consuming reactions.
6. *Enzymatic degradation* of proteins is another mechanism that is involved in the regulation of enzymatic activity (by rather drastic methods).

Posttranslational regulation mechanisms govern responses in the range of about $10^{-4} - 10^{2}$ seconds, while transcription and translation govern responses

in the range of about $10^2 - 10^8$ seconds (McAdams and Arkin, 1998) thus covering a wide range of time scale.

The above presented hierarchical control in bacteria can be influenced by mutations and (horizontal) exchange of mobile genetic elements, processes that further stress on the importance of time, a factor that really deserves special attention both in microbiology and P systems.

To illustrate the hierarchical control of a real process in bacteria, we choose the antiporter system, because the transport of ions and molecules across the cell membrane is a major topic of modern microbiology (see, e.g., Booth, 1988; Jung, 2001; Kelly and Thomas, 2001; Nicholls and Ferguson, 1992; Padan et al., 2001; Saier, 1999) and a nice example of communication between microbiology and P systems (Păun et al., 2001; Păun and Păun, 2002; Martin Vide et al., 2002; Ardelean, 2002, etc.).

$Na^+/H^+$ antiporters are membrane proteins that exchange $Na^+$ (or $Li^+$) for $H^+$ (Padan et al., 2001) that are involved in the maintenance of a relatively constant concentration (homeostasis) of these two ions inside the cell.

In *E. coli* cells which grow well in appropriate media having the pH between 6 and 8, the intracellular pH is kept in narrow limits (7.5–8.0). When growing cells are shifted from an external pH 7.2 to 8.3, the cytoplasm pH rises immediately to 8.3; however, within few minutes the cells restore the pH inside the cell to approximately the initial value. The mechanism by which *E. coli* acidifies its cytoplasm to maintain the pH value is not fully understood, but $Na^+/H^+$ antiporters play a role in this process.

Two genes encoding $Na^+$ and $Li^+$ specific antiporters were identified in *E. coli*: *nhaA* and *nhaB*. The gene *nhaA* encodes a protein that is the main antiporter which is required to withstand the upper limit concentration of $Na^+$ for growth (0.9M, pH 7.0) and to tolerate the upper pH limit for growth in the presence of $Na^+$ (0.7M NaCl, pH 8.5). The gene *nhaB* encodes a protein that acts as a housekeeper which becomes essential only in the absence of *nhaA* gene.

The experiments have shown that the increase in $Na^+$ and $Li^+$ concentrations are the environmental signals which turns on the *nhaA* gene; alkaline pH potentiates the effect of these ions but neither increase of osmolarity nor of ionic strength induces this gene. Further results showed that the intracellular $Na^+$ concentration is actually the direct signal for the transcription of the *nhaA* gene. These results demonstrate for the first time that *E. coli* has a unique regulatory network responding specifically to $Na^+$ and $Li^+$ (Padan et al., 2001).

In the presence of $Na^+$, the protein nhaR (encoded by the *nhaR* gene) undergoes a conformational change thereby inducing the transcription of the *nhaA* gene and, consecutively, the synthesis of the nhaA antiporter.

The hierarchical control in the case of $Na^+/H^+$ antiporter in *E. coli* is illustrated by the occurrence of three global regulators; one is the DNA-binding protein named *H-NS* which controls the nhaR protein and two sigma factors. One sigma factor activates promoter two (*P2*) and the other sigma factor activates promoter 1 (*P1*). *P1* is the promoter for the *nhaA* gene in cells in the exponential growth phase and its transcription is 10 times higher in $Na^+$ induced

cells than in cells grown in the absence of $Na^+$. *P2* is the main promoter for the *nhaA* gene in cells during stationary phase and is not induced by $Na^+$.

Why *P1* is the only promoter for the *nhaA* gene in cells which divide exponentially and *P2* the main promoter for cells during the stationary phase is still an open question in microbiology.

In conclusion, we suggest that the following questions deserve a further attention, believing that the advent of P systems is not only an exciting event in the field of computer science, but also an opportunity to put new questions in (micro)biology.

1. Is a happy marriage possible between the type of mathematics used in P systems and that used so far in the modelling of metabolic pathways (reviews: Bailey, 1998; Varner and Ramkhrishna ,1999; Szedlacsek, 2000; Stafford and Stephanopoulos, 2001; Wolkenhauer, 2002) or in studying genome, transcriptome, proteome, and metabolome (Spengler, 2000; Kampfner, 2002; Marijuan, 2002; Ouzounis, 2002; Gaasterland and Oprea, 2002; Goodman, 2002; see also Conrad, 1972)?
2. Would P systems theories be beneficial for microbiology, and biology in general, as a new mathematical tool to study the process of life?
3. Could the development of P systems enable the scientists to elaborate new concepts in (micro)biology and a (nondescriptive) definition of life?
4. Is the computational capacity a new aspect of the living cell? Our opinion is that the cell computes as naturally as it performs chemical reactions; to be more specific, these chemical reactions, as those shown to be involved in respiration and photosynthesis, including their coefficients, could help to find a scientific answer to this question.
5. Is the P systems paradigm the starting point for a new revolution in biology?

## References

1. P. Aldridge, K. Huges, Regulation of flagellar assembly, *Curr. Opinion Microbiol.*, 2 (2002), 160–165.
2. G. Alford, Membrane systems with heat control, *Pre-proc. Workshop on Membrane Computing*, Curtea de Argeş 2002 (G. Păun, C. Zandron, eds.), 7–15.
3. I.I. Ardelean, The relevance of cell membrane for P systems. General aspects, *Fundamenta Informaticae*, 49, 1-3 (2002), 35–43.
4. J. Bailey, Mathematical modeling and analysis in biochemical engineering: past accomplishments and future opportunities, *Biotechnol. Prog.*, 14 (1998), 8–20.
5. B.L. Bassler, How bacteria talk each other: regulation of gene expression by quorum sensing, *Curr. Opinion Microbiol.*, 2 (1999), 582–587.
6. M.G. Boersema, I.P. Solyanikova, W.J. Van Berkes, J. Vervoort, L.A. Golovleva, I.M. Rietjens, 19F NMR metabolomics for the elucidation of microbial degradation pathways of fluorophenols, *J. Ind. Microbiol. Biotechnol.*, 26 (2001), 22–34.
7. I.R. Booth, Bacterial transport: energetics and mechanisms, in *Bacterial Energy Transduction* (C. Anthony, ed.), Academic Press, London, 1988, 377–428.
8. D. Bray, Protein molecules as computational elements in living cells, *Nature*, 376 (1995), 307–312.

9. R. Bruckner, F. Titgemeyer, Carbon catabolite repression in bacteria: choice of the carbonsource and autoregulatory limitation of sugar utilisation, *FEMS Microbiol. Lett.*, 209 (2002), 141–148.
10. F.J. Bruggeman, W.C. van Heeswijk, F.C. Boogerd, H.V. Westerhoff, Macromolecular intelligence in microorganisms, *Biol. Chem.*, 9-10 (2000), 965–972.
11. M. Conrad, Information processing in molecular systems, *Curr. Modern. Biol.*, 5, 1 (1972), 1–14.
12. D.G. Davies, M.R. Parsek, J.P. Pearson, B.H. Iglewski, J.W. Costerton, E.P. Greenberg, The involvement of cell to cell signals in the development of a bacterial biofilm, *Science*, 280 (1998), 295–299.
13. F. Devaux, P. Marc, C. Jacq, Transcriptomes, transcription activators and microarrays, *FEBS Lett.*, 498 (2001), 140–144.
14. R. Freund, Energy-controlled P systems, *Pre-proc. Workshop on Membrane Computing*, Curtea de Argeş 2002 (G. Păun, C. Zandron, eds.), 221–237.
15. R. Freund, M. Oswald, GP systems with forbidding context, *Fundamenta Informaticae*, 49, 1-3 (2002), 81–102.
16. P. Frisco, H.J. Hoogeboom, Simulating counter automata by P systems with symport/antiport, *Pre-proc. Workshop on Membrane Computing*, Curtea de Argeş, 2002 (G. Păun, C. Zandron, eds.), 237–249.
17. P. Frisco, S. Ji, Towards a hierarchy of info-energy P systems, *Pre-proc. Workshop on Membrane Computing*, Curtea de Argeş, 2002 (G. Păun, C. Zandron, eds.), 265–283.
18. T. Gaasterland, M. Oprea, Whole-genome analysis: annotations and updates, *Curr. Opinion Str. Biology*, 11 (2001), 377–381.
19. N. Goodman, Biological data becomes computer literate: new advances in bioinformatics, *Curr. Opinion Biotechnol.*, 13 (2002), 68–71.
20. M.G. Gunnevijk, P.T. van den Bogaard, L.M. Venhoff, E.H. Heuberger, W.M. de Vos, M. Kleerbezem, O.P. Kuipers, B. Poolman, Hierarchical control versus autoregulation of carbohydrate utilisation in bacteria, *J. Mol. Microbiol. Biotechnol.*, 3 (2001), 401–413.
21. L.H. Hartwell, J.L. Hopfield, S. Leibler, A.W. Murray, From molecular to modular cell biology, *Nature*, 402 (1999), C47–C52.
22. K.R. Hess, W. Zhang, K.A. Baggerly, D.N. Stivers, K.R. Coombes, W. Zhang, Microarrays: handling the deluge of data and extracting reliable information, *Trends Biotechnol.*, 19 (2001), 463–468.
23. P. James, On genoms and proteoms, *Biochem. Biophys. Res. Commun.*, 231, 1 (1997), 1–6.
24. H. Jong, Modeling and simulation of genetic regulatory systems: a literature review, *J. Comput. Biol.*, 9, 1 (2002), 67–103.
25. H. Jung, Towards the molecular mechanism of Na/solute symport in prokaryotes, *Biochem. Biophys. Acta*, 1505 (2001), 131–143.
26. R.R. Kampfner, Digital and biological computing in organizations, *BioSystems*, 64 (2002), 179–188.
27. D.J. Kelly. G.H. Thomas, The tripartite ATP-independent periplasmic (TRAP) transporters of bacteria and archaea, *FEMS Microbiol. Rev.*, 25 (2001), 404–424.
28. N. Levina, S. Totemeyer, N.R. Stokes, P. Louis, M.A. Jones, I.B. Booth, Protection of Escherichia coli cells against extreme turgor by activation of MscS and MscL mechanosensitive channels: identification of genes required for MscS activity, *The EMBO J.*, 18 (1999), 1730–1737.
29. P.C. Marijuan, Bioinformation: untangling the networks of life, *BioSystems*, 64 (2002), 111–118.

30. C. Martin-Vide, A. Păun, G. Păun, G. Rozenberg, Membrane systems with coupled transport: universality and normal forms, *Fundamenta Informaticae*, 49, 1-3 (2002), 1–15.
31. H. McAdams, A. Arkin, Simulation of prokaryotic genetic circuits, *Annu. Rev. Biophys. Biomol. Struct.*, 27 (1998), 199–224.
32. F. Meng, B.J. Cargile, L.M. Miller, A.J. Forbes, J.R. Johnson, N.L Kelleher, Informatics and multiplexing of intact protein identification in bacteria and the archaea, *Nat. Biotechnol.*, 19 (2001), 952–957.
33. D.G. Nicholls, S.J. Ferguson, *Bioenergetics*, 2nd ed., Academic Press, London, 1992.
34. T.Y. Nishida, Simulation of photosynthesis by a K-subset transforming systems with membranes, *Fundamenta Informaticae*, 49, 1-3 (2002), 249–259.
35. T. Nougueira, M. Springer, Post-transcriptional control by global regulators of gene expression in bacteria, *Curr. Opinion Microbiol.*, 3 (2000), 154–158.
36. C. Ouzounis, Bioinformatics and the theoretical foundations of molecular biology, *Bioinformatics*, 18, 3 (2002), 377–378.
37. E. Padan, M. Venturi, Y. Gercham, N. Dover, Na/H antiporters, *Biochem. Biophys. Acta*, 1505 (2001), 144–157.
38. A. Păun, Membrane systems with symport/antiport: universality results, *Pre-proc. Workshop on Membrane Computing*, Curtea de Argeş, 2002 (G. Păun, C. Zandron, eds.), 333–345.
39. A. Păun, G. Păun, The power of communication: P systems with symport/antiport, *New Generation Computing*, 20, 3 (2002), 295–306.
40. A. Păun, G. Păun, A. Rodriguez-Paton, Further remarks on P systems with symport rules, *Annals of Univ. Al.I. Cuza, Iassy, Series Mathematics-Informatics*, 10 (2001), 3–18.
41. G. Păun, P systems with active membranes: Attacking NP-complete problems, *J. Automata, Languages, and Combinatorics*, 6, 1 (2001), 75–90.
42. G. Păun, From cells to computers: computing with membranes (P systems), *BioSystems*, 59 (2001), 139–158.
43. G. Păun, *Membrane computing. An Introduction*, Springer, Berlin, 2002.
44. G. Păun, Y. Suzuki, H. Tanaka, P systems with energy accounting, *Int. J. Computer Math.*, 78, 3 (2001), 343–364.
45. T.J. Phelps, A.V. Palumbo, A.S. Beliaev, Metabolomics and microarrays for improved understanding of phenotypic characteristics controlled by both genomics and environmental constraints, *Curr. Opinion Biotechnol.*, 13 (2002), 20–24.
46. L.M. Raamsdonk, B. Teusink, D. Broadhurst, N. Zhang, A. Hayes, M.C. Walsh, J.A. Berden, K.M. Brindle, D.B. Kell, J.J. Rowland, H.V. Westerhoff, K. van Dam, S.G. Oliver, A functional genomics strategy that uses metabolome data to reveal the phenotype of silent mutations, *Nat. Biotechnol.*, 19 (2001), 45–50.
47. A.J. Roe, D. Mc Laggan, C.P. O'Bryne, I.R. Booth, Rapid inactivation of the Escherichia coli Kdp K uptake system by high potassium concentration, *Mol. Microbiol.*, 35 (2000), 1235–1243.
48. M.H. Saier, Genome archeology leading to the characterization and classification of transport proteins, *Curr. Opinion Microbiol.*, 2 (1999), 555–561.
49. S.J. Singer, G.L. Nicolson, The fluid mosaic model of the structure of cell membranes, *Science*, 175 (1972), 720–731.
50. S.J. Singer, The molecular organization of membranes, *Ann. Rev. Biochem.*, 43 (1974), 805–835.
51. S.J. Spengler, Bioinformatics in the information age, *Science*, 287 (2000), 1221–1223.

52. D.E. Stafford, G. Stephanopoulos, Metabolic engineering as an integrating platform for strain development, *Curr. Opinion Microb.*, 4 (2001), 336–240.
53. G.D. Stormo, K. Tan, Mining genome data base to identify and understand new gene regulatory systems, *Curr. Opinion Microb.*, 5 (2002), 149–153.
54. J. Stulke, W. Hillen, Carbon catabolite repression in bacteria, *Curr. Opinion Microb.*, 2 (1999), 195–201.
55. S.E. Szedlacsek, Time dependent or steady-state control of metabolic systems, in *Tehcnological and Medical Implications of Metabolic Control Analysis* (A.J. Cornish-Bowden, M.L. Cardenal, eds.), Kluwer Academic Publishers, 2000, 251–258.
56. D. Thieffry, R. Thomas, Quantitative analysis of gene networks, *Pac. Symp. Biocomputing*, 1998, 77–88.
57. J. Varner, D. Ramkrishna, Mathematical models of metabolic pathways, *Curr. Opinion Biotechnol.*, 10 (1999), 146–150.
58. M.P. Washburn, J.R. Yates, Analysis of the microbial proteome, *Curr. Opinion Microbiol.*, 3 (2000), 292–297.
59. R.A. van Bogelen, K.D. Greis, R.M. Blumenthal, T. Tani, R.G. Matthews, Mapping regulatory networks in microbial cells, *Trend. Microbiol.*, 7, 8 (1999), 320–328.
60. O. Wolkenhauer, Mathematical modelling in the post-genome era: understanding genome expression and regulation: a system theoretic approach, *BioSystems*, 65 (2002), 1–18.

# A Software Simulation of Transition P Systems in Haskell*

Fernando Arroyo[1], Carmen Luengo[1], Angel V. Baranda[2], and Luis de Mingo[3]

[1] Dept. Lenguajes, Proyectos y Sistemas Informáticos
Escuela de Informática, Universidad Politécnica de Madrid
Crta. de Valencia km. 7, 28031 Madrid, Spain
{farroyo,cluengo}@eui.upm.es
http://www.lpsi.eui.upm.es
[2] Dept. Inteligencia Artificial
Facultad de Informática, Universidad Politécnica de Madrid
Campus de Montegancedo, Boadilla del Monte, 28660 Madrid, Spain
http://www.dia.fi.upm.es
[3] Dept. Organización y Estructura de la Información
Escuela Universitaria de Informática, Universidad Politécnica de Madrid
Crta. de Valencia Km. 7, 28031 Madrid, Spain
lfmingo@eui.upm.es
http://www.oei.eui.upm.es

**Abstract.** P systems are a parallel and distributed computational model, based on the membrane structure notion. Membranes define regions. Inside regions, objects and rules are placed in order to make evolve the P system. Evolution is achieved by transitions between two consecutive system configurations. Therefore, a computation can be obtained as a transitions series between consecutive configurations. Where and how P systems can be implemented is nowadays an open problem, but implementation on digital computers could be one way to show the capabilities of such systems. This paper presents a transition P systems implementation in Haskell, based on a theoretical framework previously developed.

## 1 Introduction

Transition P systems were introduced by G. Păun [5]. They are the simplest variant of P systems, however, they have the essential components of many variants of P systems. In fact, in any variant of P systems two basic components can be found: the static structure and the dynamic structure. The static structure consists of the membrane structure and the multisets or strings associated with each region defined by the membrane structure, while the dynamic structure consists of the rules associated with the regions defined by the membrane structure of the P system. These rules are responsible for the P system evolution, making changes in the static structure, changing the multisets of strings associated with

* Work partially supported by contribution of EU commission under The Fifth Framework Programme, project "MolCoNet" IST-2001-32008.

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 19–32, 2003.

each region, or even changing the membrane structure. Rules can be defined in many different ways, depending on the variant of P system considered.

One of the main goals of the implementation described here is to keep close enough the software to the philosophy of membrane computing. We try to achieve this, keeping in the different software modules the fundamental functionality of the corresponding ingredients of P systems. Therefore, any item (multiset, rule, region, membrane, etc) has its corresponding module in the developed software. Moreover, the functionality that the module provides to the software system is what it is expected from it. The static and dynamic structures of P systems have also their corresponding projection on the software.

The following sections are devoted to explain the software architecture in connection with a previous theoretical work of formalization corresponding to transition P systems. The formalization work has been essential in order to analyze and design the software architecture.

## 2 System Architecture Description

The current implementation of transition P system has been written in Haskell. The language has been chosen in order to keep as close as possible the implementation to a previously developed theoretical work, see [4],[3],[1] and [2]. Here we describe the software architecture of the implementation, giving a short description of the main software modules (implementing Abstract Data Types) and their functional dependencies.

### 2.1 Abstract Data Type Multiset

This module, ADTMultiset, implement the multiset concept. The module provides the Haskell type Multiset defined as follows:

- data Multiset a = CMultiset [(a, Int)]

From the above data definition in Haskell, multisets have been defined as lists of tuples with two elements: the first one is a char and the second one is the number of copies of the first element in the multiset.
This module also provides the most commonly used function in transition P system over multisets. The following list shows the type of such functions in Haskell:

- Empty multiset:
  **isEmptyMS** :: Multiset a → Bool
- Multiset Union
  **(.‖:++:‖.)** :: Eq a => Multiset a → Multiset a → Multiset a
- Multiset Intersection
  **(.‖:\/:‖.)** :: Eq a => Multiset a → Multiset a → Multiset a
- Multiset Difference
  **(.‖:\\:‖.)** :: Eq a => Multiset a → Multiset a → Multiset a

- Multiset Size
  **SizeMS** :: Multiset a → Int
- Support Multiset
  **supportMS** :: Multiset a → [a]

For a formal definition of these functions over Multisets we refer to [4]. The module implements a set of auxiliary functions that complements the Abstract Data Type, and the input/output functions for Multiset.

### 2.2 Abstract Data Type Relationship

This module, ADTRelationship, implements in Haskell a general binary relationship defined over rules. It is used for implementing the priority relation, which is used by the evolution rule module. The module provides all the needed functionality for handling any partial order relation defined over a given arbitrary finite set, including the finding of the maximal and minimal set over the relation, and its transitive closure.

The module defines the Haskell data type Relationship as follows:

- data Relationship a b = CRelationship [(a, b)]

In this case, the type Relationship has been defined as a list of tuples with two elements. The relation has been considered as a subset of a general Cartesian product of two arbitrary sets.

The Haskell type of the most relevant functions from the ADTRelationship module are the following:

- **belonging: (..|:=>.|..)**
  (..|:=>.|..) :: (Eq a, Eq b) => Relationship a b → (a, b) → Bool
- **imageR**
  imageR :: (Eq a, Eq b) => Relationship a b → [b]
- **maximalR**
  maximalR :: Eq a => Relationship a a → [a] → [a]
- **minimalR**
  minimalR :: Eq a => Relationship a a → [a] → [a]
- **transitiveClosureR**
  transitiveClosureR :: Eq a => Relationship a a → Relationship a a

### 2.3 Abstract Data Type Multirelationship

During the software development, some very interesting similarities between Relationship and the consequent of one evolution rule were found. These similarities were responsible for the inclusion of the ADTMultirelationship module into the software. It can be said that a relationship is to a set what a multirelationship is to a multiset. If one looks carefully to ADTRelationship and ADTMultirelationship modules, one can see that all the implemented functions on the first one can be implemented on the second one.

This module defines the Haskell data type Multirelationship as follows:

- data Multirelationship a b = CMultirelationship [((a, b), Int)]

The module imports two of the modules defined above:

- import ADTMultiset
- import ADTRelationship

These import clauses determine the functional dependency of the module on the previously defined modules.

The Haskell type of the most relevant functions defined in this module according to the software implementation are:

- **Support**
  supportMR :: (Eq a, Eq b) => Multirelationship a b → Relationship a b
- **Origin From**
  originFromMR :: (Eq a, Eq b) => b → Multirelationship a b → Multiset a
- **Multirelationship Union**
  (..‖:++:‖..) :: (Eq a, Eq b) => Multirelationship a b →
  Multirelationship a b → Multirelationship a b

### 2.4 Abstract Data Type Rule

This module implements evolution rules in transition P systems. The functionality of the module is based on a previous theoretical work, see [4], [3], [1] and [2].

The ADTRule module defines the data type Rule as follow:

- data Rule = CRule Int (Multiset Char) (Multirelationship Char Int) Bool

A rule is a tuple with four elements; the first one is a label for the rule, the second one is the antecedent, the third one is the consequent and the fourth one is the dissolution capability of the rule.

The module import the following previously defined modules:

- import ADTRelationship
- import ADTMultiset
- import ADTMultirelationship

These import clauses determine the functional dependency of the module on the previously defined modules ADTRelationship, ADTMultiset and ADTMultirelationship.

The most relevant functions defined in this module are the following:

- **LabelRule**
  LabelRule :: Rule → Int
- **inputRule**
  inputRule :: Rule → Multiset Char

- **outputRule**
  outputRule :: Rule → Multirelationship Char Int
- **dissolvesRule**
  dissolvesRule :: Rule → Bool
- **outputToRule**
  outputToRule :: Int → Rule → Multiset Char
- **isUssefulRule**
  isUssefulRule :: [Int] → Rule → Bool
- **isApplicableRule**
  isApplicableRule :: Multiset Char → Rule → Bool
- **isMinorRuleIn**
  isMinorRuleIn :: Relationship Int Int → (Rule, Rule) → Bool
- **addRule**
  addRule :: Rule → Rule → Rule

### 2.5 Abstract Data Type Region

From the computational point of view, the regions are considered as bags where multisets, rules, and a priority relation among rules are placed.

The Haskell module ADTRegion defines the data type Region as follows:

- data Region = CRegion (Multiset Char) [Rule] (Relationship Int Int)

Hence, a region is defined as a tuple with three elements, the first one is an object multiset, the second one is a set of evolution rules and the third one is a priority relation defined among rules. The priority relation could be the empty relation.

The functional dependencies of this module with others ones are expressed by the following import clauses:

- import ADTMultiset
- import ADTRelationship
- import ADTRule

The Haskell data type of most important functions of the module are:

- contentRegion :: Region → Multiset Char
- rulesRegion :: Region → [Rule]
- prioritiesRegion :: Region → Relationship Int Int
- reactionRegion :: StdGen → [Int] → Region → Rule

As it can be seen from the list of functions defined above, the module provides functions in order to obtain all the elements of the tuple: object multiset, set of rules and priority relation. Region module is responsible for computing one rule able to make evolve the region in one step. This functionality is implemented in the reaction function.

The reaction function computes in a random way a complete linear combination of evolution rules. This means that no other applicable evolution rule can be included in the linear combination. For more information about complete multisets of evolution rules and their corresponding linear combinations we refer to [1].

### 2.6 Abstract Data Types Innermembrane and Externmembrane

These two modules, are quite similar. In fact, they have the same number of functions and the same functionality. However, there is one important difference between them. The external membrane cannot be dissolved, while inner membranes can be dissolved. This fact is very important when one wants to implement evolution in transition P systems. Inner membranes can evolve in two ways: dissolving or not dissolving the membrane; the external membrane evolves in one way, the dissolution is not allowed. This is the reason why two different modules have been implemented in the software. For a more comprehensive explanation of evolution in transition P systems, we refer to [1].

The Haskell modules ADTInnermembrane and ADTExternalmembrane implement the data type Innermembrane and Externalmembrane respectively. The data type definitions in both modules are:

- data Innermembrane = CInnermembrane Int Region [Innermemb.]
- data Externalmembrane = CExternalmembrane Int Region [Innermemb.]

The data type definitions are very close because there is only one external membrane in P systems, the *"skin"*, and all the other membranes are inner to it.

Membranes are defined as tuples with three elements: the first one is the label associated to the membrane, the second one is the region associated to the membrane (the bag) and the third one is the set of inner membranes. This set of inner membranes can be the empty set and then, the membrane is said to be elementary.

The functional dependencies of these modules are defined by the following import clauses:

- ADTExternalmembrane
  - import ADTMultiset
  - import ADTRelationship
  - import ADTInnermembrane
  - import DATRegion
  - import ADTRule
- ADTInnermembrane
  - import ADTMultiset
  - import ADTRelationship
  - import DATRegion
  - import ADTRule

The functionality of these modules are the following:

- ADTExternalmembrane
  - labelEM :: Externalmembrane → Int
  - regionEM :: Externalmembrane → Region
  - innerMembranesEM :: Externalmembrane → [Innermembrane]

  - contetEM :: Externalmembrane → Multiset Char
  - rulesEM :: Externalmembrane → [Rule]
  - prioritiesEM :: Externalmembrane → Relationship Int Int
  - innerLabelsEM :: Externalmembrane → [Int]
  - reactionEM :: StdGen → Externalmembrane → Rule
  - evolutionEM :: StdGen → Multiset Char → Externalmembrane → ([Innermembrane], Multiset Char)
- ADTInnermembrane
  - labelIM :: Innermembrane → Int
  - regionIM :: Innermembrane → Region
  - innerMembranesIM :: Innermembrane → [Innermembrane]
  - contetIM :: Innermembrane → Multiset Char
  - rulesIM :: Innermembrane → [Rule]
  - prioritiesIM :: Innermembrane → Relationship Int Int
  - innerLabelsIM :: Innermembrane → [Int]
  - reactionIM :: StdGen → Innermembrane → Rule
  - evolutionIM :: StdGen → Multiset Char → Innermembrane → ([Innermembrane], Multiset Char)

Probably, the most important functions in both modules are those related to evolution in membranes, and hence in P systems evolution. Functions involved in evolution are named in two ways: reaction and evolution.

A reaction function obtains from the associated region one evolution rule able to make evolve the membrane in one step. Once the complete linear combination of evolution rules has been obtained, the evolution is produced inside the membrane.

### 2.7 Abstract Data Type PSystem

This Haskell module implements a transition P system. Basically, it provides the abstract data type PSystem as one output object multiset and one external membrane. The output object multiset will be the external output of the P system, if there is any.

The module defines the data type Psystem as follows:

- data PSystem = CPSystem (Multiset Char) Externalmembrane

The functional dependencies are given by the following import clauses:

- import ADTMultiset
- import ADTExternalmembrane

The most important function on this module is:

- evolutionPS :: StdGen → PSystem → PSystem

This function make evolve the transition P system from one initial configuration to the next one applying evolution to the external membrane. The evolution is obtained by a random generation of one complete multiset of evolution rules. Moreover, evolution only gives one evolution step. Hence, it will be necessary to apply several times the function in order to get a successful computation.

Figure 1 shows the functional dependencies among the main software modules described until now.

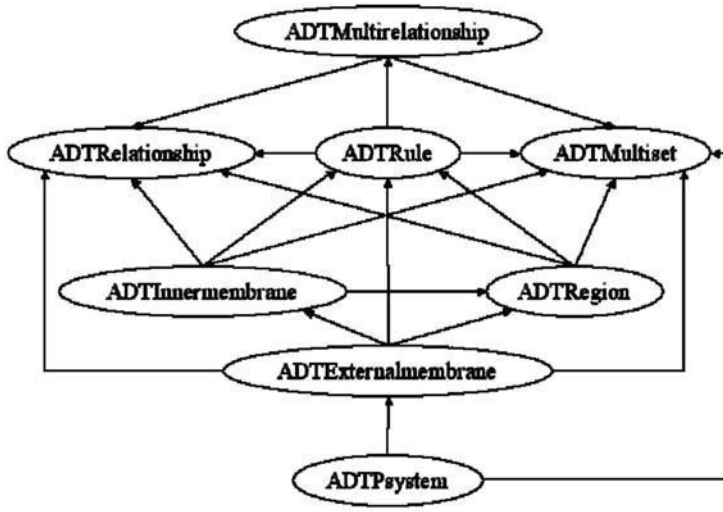



**Fig. 1.** Functional Dependencies Among Main Software Modules

### 2.8 The Main Module

This Haskell module implements the external function that makes possible to launch the program execution. The module imports the following modules:

- import DATPSystem
- import DATMultiset
- import DATExternalMembrane

The functionality of this module is to make evolve several times the transition P system in order to get a successful computation. The input to this function is a text file on which the initial configuration of the P system is described (see Section 3.1 of this paper). The output produced by the program is a new text file in which the new configuration is stored. This output file must be used as a new input to the program in order to get one more evolution step of the P system.

## 3 An Example of Use

This section will describe an example of use of the transition P systems software simulator. In order to check the code execution of our software we have taken from [5] the first example of Section 6. It will be recalled below for illustrating the input file definition for our program.

Consider the P system of degree 4

$$
\begin{aligned}
\Pi &= (V, \mu, \omega_1, ..., \omega_4, (R_1, \rho_1), ..., (R_4, \rho_4), 4) \\
V &= \{a, b, c, d, e\} \\
\mu &= [_1[_2[_3]_3[_4]_4]_2]_1 \\
\omega_1 &= \lambda, R_1 = \emptyset, \rho_1 = \emptyset \\
\omega_2 &= \lambda, R_2 = \{r_1 : a \to b, r_2 : b \to b(c, in_4), r_3 : d^2 \to de, r_4 : d \to e\delta\}, \\
&\quad \rho_2 = \{r_4 < r_3\} \\
\omega_3 &= \{de\}, R_3 = \{r_1 : d \to d^2, r_2 : e \to ea, r_3 : e \to a\delta\}, \rho_3 = \emptyset \\
\omega_4 &= \lambda, R_4 = \emptyset, \rho_4 = \emptyset
\end{aligned} \tag{1}
$$

As shown in [5], the set of numbers generated by $\Pi$ is:

$$N(\Pi) = \{m^2 \mid m \geq 1\}. \tag{2}$$

First of all, the initial configuration (1) must be translated to an input file for the software simulator. The explanation of how this translation is done is given in the next subsection.

### 3.1 Structure of the Input File

This section shows the input file containing the initial configuration of the P system $\Pi$ described above. Moreover, it explains the way to make the translation from the formal definition of the P system to an input file. The corresponding input file for $\Pi$ will be given below.

The first line of the input file is the object multiset that the P system outputs after each evolution step. The second line is just the beginning of the *"skin"* membrane and, hence, the beginning of the membrane structure of the P system. All others membranes are included in it, as it was described in the Haskell modules ADTExternalmembrane and ADTInnermembrane.

The membrane structure is defined as usual in P systems. Between two square brackets $[_n \ldots ]_n$, we will place all the membrane elements in the following order:

1. objects multiset
2. set of evolution rules
3. priority relation among rules
4. set of membranes inner to the membrane we are defining with all theirs elements (e.g., 1, 2, 3, 4).

The following subsections will describe the different elements of the input file.

**Example of an Input File.** This subsection shows the input file for the P system $\Pi$ previously considered. The file is shown without any comments in order to appreciate the whole structure of it.

```
{}
[ 1
   {}
   {}
   {}
   [ 2
      {}
      {r 1 : {('a', 1)} ---> {(('b', 0), 1)},
       r 2 : {('b', 1)} ---> {(('b', 0), 1), (('c', 4), 1)},
       r 3 : {('d', 2)} ---> {(('d', 0), 1), (('e', 0), 1)},
       r 4 : {('d', 1)} --<> {(('e', 0), 1)}}
      {(4, 3)}
      [ 3
         {('e', 1), ('a', 1), ('d', 2)}
         {r 1 : {('d', 1)} ---> {(('d', 0), 2)},
          r 2 : {('e', 1)} ---> {(('e', 0), 1), (('a', 0), 1)},
          r 3 : {('e', 1)} --<> {(('a', 0), 1)}}
         {}
      ] 3
      [ 4
         {}
         {}
         {}
      ] 4
   ] 2
] 1
```

Now, some comments have been added to the file, marked with (* *), to explain some details.

```
{} (*Output multiset from the P system*)
[ 1 (*Beginning of the
"skin" membrane*)
   {}   (*Multiset on the ‘‘skin" membrane*)
   {}   (*Set of rules associated to the ‘‘skin" membrane*)
   {}   (*Priority relationship among rules on
                          the ‘‘skin" membrane*)
   [ 2  (*Beginning of the inner membrane 2*)
      {}    (*Multiset of the membrane 2*)
      {(*Set of rules associated to membrane 2*)
       r 1 : {('a', 1)} ---> {(('b', 0), 1)},
       r 2 : {('b', 1)} ---> {(('b', 0), 1), (('c', 4), 1)},
       r 3 : {('d', 2)} ---> {(('d', 0), 1), (('e', 0), 1)},
```

```
      (*The rule r 4 dissolves the membrane if it is applied*)
      r 4 : {('d', 1)} --<> {(('e', 0), 1)}}
    (*Priority relationship among rules on the membrane 2*)
    {(4, 3)}
    [ 3 (*Beginning of the inner membrane 3*)
       (*Multiset of the membrane 3*)
       {('e', 1), ('a', 1), ('d', 2)}
       {(*Set of rules associated to the inner membrane 3*)
        r 1 : {('d', 1)} ---> {(('d', 0), 2)},
        r 2 : {('e', 1)} ---> {(('e', 0), 1), (('a', 0), 1)},
        (*The rule r 3 dissolves the membrane if it is applied*)
        r 3 : {('e', 1)} --<> {(('a', 0), 1)}}
        (*Priority relationship among rules on the membrane 3*)
       {}
    ] 3   (*End of the inner membrane 3*)
    [ 4   (*Beginning of the inner membrane 4*)
       {} (Multiset of the inner membrane 4*)
       {} (*Set of rules associated to the membrane 4*)
       (*Priority relationship among rules on the membrane 4*)
       {}
    ] 4   (*End of the inner membrane 4*)
  ] 2  (*End of the inner membrane 2*)
] 1 (*End of the skin membrane. End of P system *)
```

How it can be appreciated from the input file structure, translation from the classical formal definition of a transition P system to the file is not very difficult. We only need to rearrange the data into the file in order to have the input to the program.

Now, we will explain the way to define the different components of membranes in the input file.

**Multiset of Objects.** A multiset of objects is described in the input file as a set of tuples with two elements, the first one is an element from the alphabet and the second one is the multiplicity of the object in the multiset. For instance, the object multiset $a^2bd$ is described on the input file as the set:

```
{('a', 2), ('b', 1), ('d',1)}
```

**Rules Description.** In transition P systems, the rules have three components: the antecedent, the consequent and the dissolving capability.

Now we will describe step by step all these elements in order to correctly define rules in the input file.

First of all, in order to implement the priority relation among rules, all the rules of the membrane must be labelled. The label starts with a character followed by a blank, a different number for each rule in the membrane, a blank and

the character ':'. In fact, we only used the number associated to each rule for defining the priority relation.

The antecedent is a multiset of objects as described above.

The consequent has been considered here as a multirelationship, therefore, it has a special description in the input file. Basically, the consequent is described as a tuple with two elements, the first one is a tuple containing the object and the targeting address, where the rule is sending the object; the second one is the number of copies of the object that the rule is sending to the target membrane.

Targeting is here described by numbers:

- **-1** the copies of the object are sent outside of the membrane.
- **0** the copies of the object are preserved in the membrane
- **n** the copies of the object are sent to the membrane with label $n$.

Finally, the dissolving capability of the rule is included in the symbol representing the arrow of the rule.

The following example will clarify the way of defining rules in the input file. Consider the following two rules:

$$r_1 : b \to b(c, in_4) \tag{3}$$

$$r_2 : d \to e\delta \tag{4}$$

The rule (3) will not dissolve the membrane, while the rule (4) will do. Their representation in the input file are the following:

```
r 1 : {('b', 1)} ---> {(('b', 0), 1), (('c', 4), 1)}
r 2 : {('d', 1)} --<> {(('e', 0), 1)}
```

Note the symbol representing the arrow in rule r 2. This symbol represents the dissolution capability that the rule has.

We must also note that rules are included in regions arranged in sets. Therefore, if the membrane contains the two rules described above, the set of rules are included between { } separated by commas, hence, the representation in the input file will be:

```
{r 1 : {('b', 1)} ---> {(('b', 0), 1), (('c', 4), 1)},
 r 2 : {('d', 1)} --<> {(('e', 0), 1)}}
```

**Priority Relation Description.** The last element included in a membrane, before the set of its inner membranes, is the priority relation among rules. This relation is described in the input file as a set of tuples with two natural numbers. Each number in the tuple is the number associated to the label of a rule from the membrane. The first element of the tuple has a lower priority than the second one.

The Haskell module ADTRelationship calculates the transitive closure for the set described in the input file. Therefore, we only need to express the priorities among rules as it is usual in transition P systems.

If we consider now the following priority relation among rules (3) and (4) {r 1 > r 2}, the input file will contain the set:

```
{(2, 1)}
```

**Inner Membranes Description.** Membranes inner to a given membrane must be placed inside it. The definition of inner membranes follow the same principles explained until now.

### 3.2 Running the Program

This section is related to software execution. The software has been written in Haskell, a functional language, and the chosen interpreter has been Hugs98 for Microsoft Windows. The interpreter for other operating systems can be found at: `http://cvs.haskell.org/Hugs/pages/downloading.htm`. The source code can also be downloaded from `http://psystems.disco.unimib.it`, the P systems web page.

Once we have installed the Hugs98 interpreter, we must follow the steps:

1. Start the Hugs98 interpreter.
2. Open the "*Principal*" Haskell module
3. Type *"main"* + <*return*> (this will launch the program).
4. The program will ask for a file (containing the transition P system). Type the name of input file and return.
5. Then it will ask for an output file; type the appropriate name for the output.
6. The program will make evolve only one evolution step and stops. One can make evolve again the transition P system another evolution step going to the step #3 and typing the output file of the #5 as the new input file for the next iteration.

## 4 Conclusions

This paper shows how a software simulation of transition P systems can be implemented with some constrains, mainly those due to their inherent parallelism and non-determinism. The presented implementation is very close to the general theory of transition P systems, and it does not present many difficulties in order to translate transition P systems to an input file in order to start the software execution.

The P system can be easily modified in the input file. The software gives an additional facility to the user in order to change the static and dynamic structure of the P system and drive the execution of the computational device in the right way.

We believe that simulations of P systems on digital computers is a good alternative to the *in vitro* implementation as long as we have no other support (for instance, special hardware devices for executing P systems) where P systems can be executed.

## References

1. F. Arroyo, A.V. Baranda, J. Castellanos, C. Luengo, L.F. Mingo, A Recursive Algorithm for Describing Evolution in Transition P Systems, Pre-Proceedings of Workshop on Membrane Computing, Curtea de Arges, Romania, August 2001, Technical Report 17/01 of Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain, 2001, 19–30.
2. F. Arroyo, A.V. Baranda, J. Castellanos, C. Luengo, L.F. Mingo, Structures and Bio-Language to Simulate Transition P Systems on Digital Computers, in Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View (C.S. Calude, Gh. Paun, G. Rozenberg, A. Salomaa, eds.), Lecture Notes in Computer Science 2235, Springer-Verlag, 2001, 1–16.
3. A.V. Baranda, J. Castellanos, F. Arroyo, R. Gonzalo, Towards an Electronic Implementation of Membrane Computing: A Formal Description of Nondeterministic Evolution in Transition P Systems, Proc. 7th Intern. Meeting on DNA Based Computers (N. Jonoska, N.C. Seeman, eds.), Tampa, Florida, USA, 2001, 273–282.
4. A.V. Baranda, J. Castellanos, R. Gonzalo, F. Arroyo, L.F. Mingo, Data Structures for Implementing Transition P Systems in Silico, Romanian J. of Information Science and Technology, 4, 1-2 (2001), 21–32
5. G. Păun, Computing with Membranes, Journal of Computer and Systems Sciences, 61, 1 (2000) 108–143.

# Authentication of Messages Using P Systems

Adrian Atanasiu

Faculty of Mathematics, Bucharest University
Str. Academei 14, sector 1
70109 Bucharest, Romania
aadrian@pcnet.ro

**Abstract.** The paper is an attempt to use P systems in dealing with a cryptographic issue, that of message authentication. Two algorithms are proposed, with and without confirmation from the sender, based on P systems with active membranes. We are not concerned with the practical usefulness of these algorithms, but with proving the usefulness of the membrane computing framework in addressing the authentication question.

## 1 Prerequisites

For the basic notions, notations, and results about P systems we refer to [5], [7], [9]. In this paper we use a variant of P systems with active membranes similar to the one defined and used in [2] and [3].

For the elements of cryptography, authentication, and electronic signature used here we refer to [6], [11], and [12], and for formal language elements we refer to [10].

A *P system with active membranes* is a construct $\Pi = (V, T, H, \mu, w_0, w_1, \ldots, w_n, R)$, where:

1. $n \geq 1$;
2. $V$ is an alphabet (the total alphabet of the system); the strings from $V^*$ are called *objects*;
3. $T \subseteq V$ (the terminal alphabet);
4. $H$ is a finite set of labels for membranes;
5. $\mu$ is a *membrane structure*, consisting of $m$ membranes, labeled (not necessarily in a one-to-one manner) with elements of $H$; there is a (unique) membrane labelled with 0, called *skin*; all the other membranes are inside of the skin; the membranes can be neutral or polarized (positive or negative);
6. $w_0, w_1, \ldots, w_n$ are strings over $V$, placed initially in the $n+1$ regions of $\mu$;
7. $R$ is a finite set of *development rules* of the following types:
   (a) $[_i x \to y]_i^\alpha, \quad x \in V^+,\ y \in V^*,\ \alpha \in \{0, +, -\}$.
   This is an internal rule, having no effect outside the membrane $i$ or on its polarity.
   (b) $[_i x]_i^\alpha \to y[_i\ ]_i^\beta, \quad x \in V^+, y \in V^*,\ \alpha, \beta \in \{0, +, -\}$.
   A string can go out of a membrane, possibly transformed into another string.

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 33–42, 2003.

(c) $x[_i]_i^\alpha \to [_i y]_i^\beta, \quad x \in V^+,\ y \in V^*,\ \alpha, \beta \in \{0, +, -\}$.
A string can go into a membrane, possibly transformed into another string.

(d) $[_i x]_i^\alpha \to y, \quad x \in V^+, y \in V^*, \quad \alpha \in \{0, +, -\},\ i \neq 0$.
The dissolution of a membrane. The skin is never dissolved.

(e) $[_i x]_i^\alpha \to [_i y]_i^\beta [_i z]_i^\beta, \quad x \in V^+,\ y, z \in V^*,\ \alpha, \beta \in \{0, +, -\}, \quad i \neq 0$.
The division of membranes: a membrane can be duplicated in two membranes with the same label. The skin is never divided.

(f) $[_i x]_i^+ [_j y]_j^\alpha \to [_i x [_j w]_j^\alpha]_i^+, \quad y \in V^+, x, w \in V^*,\ \alpha \in \{0, -\}$.
The subordonation, which models the biological operation of *endocytosis*. The string $x$ (if appears) controls this operation and plays the role of a catalyst.

These rules are applied according to the following principles ([8]):

- All the operations are applied in parallel to all nonempty strings and all membranes to which they can be applied. At one step, a string or a membrane can be used by only one operation, non-deterministically chosen (if there is no priority relation used), but any string or membrane which can evolve by an operation of any form, should evolve.
- If a membrane is dissolved, then all the strings and membranes situated in its region are left free in the region immediately above it.
- All the strings and membranes not involved in the transition do not evolve, and they are passed unchanged to the next step.

Note that the skin can use only rules of types $(a)$ and $(b)$. The strings which are sent outside the skin form the output of the P system.

## 2 Authentication of Messages without Confirmation

Authentication is a widely used term in the process of transmission of messages, being one of the most important information security objectives. Generally, an identity authentication technique assures one communication partner (usually called *Alice*) of the identity of a second partner (called *Bob*). This authentication can be mutual or unilateral (if *Bob* whishes or not to be sure that *Alice* was the right receiver of the message).

The 'bad' person is often called *Oscar*; *Oscar* intercepts the messages, tries to read them, tries to modify them, and tries to convince *Alice* that *Bob* is the sender of these modified messages.

The two authentication protocols constructed here are based on the idea of Merkle ([6]) to use a tree structure of information.

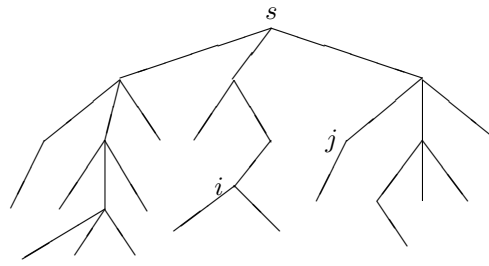
Thus, we shall consider the membrane structure $\Pi$ as a labeled tree, having the property that there are two strings $w_i, w_j$ situated in two distinct membranes, so that the message $m$ is a substring of $w_i$ and the sender authentication data $ID$ forms a substring of $w_j$.

Obviously, $m, ID \in T^+$.

In this context, a P system with active membranes as defined above is called a *comunication membrane system*, in short, a CMS.

Both specified information (message and identification data) will be considered here as sequences whose content will be ignored; so, they can be considered as *objects*.

The idea is that both the message and the identification data can be read when they are outside the skin, and the problem is to find them in the "cell" and to bring them out without changing the system in an "illegal" (and noticeable) manner.




Assume that the membrane $i$ (here the label $i$ concerns the position, not the membrane label, because the system can contain several membranes labeled with $i$) contains the message $m_{Bob}$ (written as a string of characters), and the membrane $j$ contains $ID_{Bob}$ (a string of characters identifying *Bob*: name, birth data, social security number etc).

Moreover, $ID_{Bob}$ contains an address of *Bob*.

In the first algorithm (authentication of the message $m_{Bob}$ without confirmation), we assume that *Alice* and *Bob* have two "identification enzymes" $\alpha_{Alice}$ and respectively $\alpha_{Bob}$; these objects are produced by these two partners and transmitted before exchanging the messages.

*Example 1.* Let us consider that *Alice* is a bank, and *Bob* is a beneficiary (user). When *Bob* opens an account at the bank, he offers for its own identification $\alpha_{Bob}$ (this string can have nothing in common with Bob's name, address etc); the bank can also produce such an object for *Bob* (this is a problem of trustness between these two partners).

Also, the bank gives *Bob* its own "key" $\alpha_{Alice}$ (this can be common for several users).

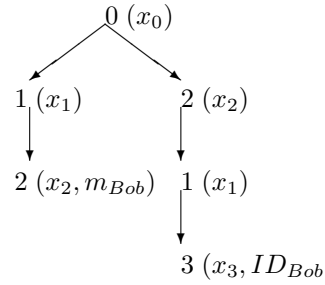
Let us suppose that *Alice* receives a $CMS\ \Pi$ and wants to read the message $m_{Bob}$ (signed by *Bob*). She will follow the following protocol:

1. *Alice* introduces in $\Pi$ the object $\alpha_{Alice}$; it will generate the following actions:
   (a) Finds a path to the node $j$;
   (b) Takes outside the skin the sequence $ID_{Bob}$ (possibly a copy) contained by this node;
   (c) Transforms the $CMS\ \Pi$ into a P system $\Pi'$ (possibly a $CMS$, too).

   In this way *Alice* finds whether $\Pi$ was previously attacked by *Oscar*.

2. *Alice* identifies *Bob* (using $ID_{Bob}$); then she selects and introduces the object $\alpha_{Bob}$ in $\Pi'$. This object:
   (a) Finds the path to the node $i$ (if $\alpha_{Bob}$ is the correct "enzyme");
   (b) Takes outside the skin the message $m_{Bob}$ (possibly a copy);
   (c) Modifies the P system $\Pi'$;
   As a variant, after this protocol is finished, the message $m_{Bob}$ can be destroyed in $\Pi'$.

*Example 2.* Let us consider the $CMS$ defined by the tree-structure:




Here:

- $x_0, x_2, x_3 \in V \setminus T$ are "jailer" objects, which start the destroying of the $CMS$ $\Pi$ if they detect the illegal intrusion of any object;
- $m_{Bob} \in V^+$ is a sequence which represents the message sent by *Bob*;
- $ID_{Bob} \in V^+$ is a sequence which identifies the sender.
- For intermediate transformations also some objects $\alpha_i, \beta_i \in V \setminus T$ $0 \leq i \leq 3$, are used.

Thus, *Bob* sends to *Alice* the $CMS$ $\Pi$ with the initial configuration

$$[_0[_1x_1[_2x_2, m_{Bob}]_2^0]_1^0[_2x_2[_1x_1[_3x_3, ID_{Bob}]_3^0]_1^0]_2^0]_0^0.$$

The development rules of $\Pi$ are:

1. (a) $\alpha_{Alice}[_0]_0^0 \to [_0\alpha_0]_0^-$,
   (b) $\alpha_{Bob}[_0]_0^0 \to [_0\beta_0]_0^+$
   (the rules for starting the protocols of finding the sender $ID_{Bob}$ and respectively the mesage $m_{Bob}$).
2. (a) $\alpha_i[_j]_j^0 \to [_j\alpha_j]_j^-$, where $(i,j) \in \{(0,2),(1,3),(2,1)\}$ (the legal path to the sender identification data);
   (b) $\beta_i[_j]_j^x \to [_j\beta_j]_j^+$, where $(i,j) \in \{(0,2),(1,2),(2,1)\}$, $x \in \{0,+\}$ (the legal path to the message $m_{Bob}$).
3. (a) $[_i\alpha_i x_i \to \alpha_i]_i^-$, $\quad 0 \leq i \leq 3$,
   (b) $[_i\beta_i x_i \to \beta_i]_i^+$, $\quad 0 \leq i \leq 2$;
   (the "jailer" objects confirm the receiving of correct enzymes);
4. (a) $[_i\alpha x_i \to x_i]_i^-$, $\quad$ for all $\alpha \neq \alpha_i$,

(b) $[_i \alpha x_i \to x_i]_i^+$, for all $\alpha \neq \beta_i$
(any illegal object is destroyed by the "jailers"). These rules have top priority in the P system.

5. (a) $[_i ID_{Bob}]_i^- \to ID_{Bob}[_i]_i^0$, $0 \leq i \leq 3$ (the negative polarity creates a path for the sequence $ID_{Bob}$ in order to go out from the $CMS\ \Pi$).
   (b) $[_i m_{Bob}]_i^+ \to m_{Bob}[_i]_i^0$, $0 \leq i \leq 3$ (the positive polarity creates a path for the message $m_{Bob}$ in order to go out from the system $\Pi'$)
   (after these operations are over, the polarity becomes neutral).
6. $[_3 \alpha_3]_3^0 \to \alpha_1$, $[_1 \alpha_1]_1^0 \to \alpha_2$, $[_2 \alpha_2]_2^0 \to [_2]_2^+$, $[_2]_2^+ [_1]_1^0 \to [_2 [_1]_1^0]_2^+$ (development rules which modify the $CMS\ \Pi$, a new P system $\Pi'$ being obtained).

Using development rules $(1a) - (4a)$, the $CMS\ \Pi$ reaches the configuration

$$[_0 [_1 x_1 [_2 x_2, m_{Bob}]_2^0]_1^0 [_2 [_1 [_3 \alpha_3, ID_{Bob}]_3^-]_1^-]_2^-]_0^-$$

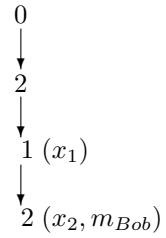
With rules $(5a)$ the configuration obtained is

$$ID_{Bob}[_0 [_1 x_1 [_2 x_2, m_{Bob}]_2^0]_1^0 [_2 [_1 [_3 \alpha_3]_3^0]_1^0]_2^0]_0^0.$$

The rules (6) transform this configuration in

$$ID_{Bob}[_0 [_2 [_1 x_1 [_2 x_2, m_{Bob}]_2^0]_1^0]_2^+]_0^0,$$

represented by the following tree (the polarities of the nodes and the sequence $ID_{Bob}$ obtained outside the skin are ignored):




If *Alice* introduces the object $\alpha_{Bob}$ in this configuration, the development rules $(1b) - (5b)$ lead to the final result

$$m_{Bob}[_0 [_2 [_1 [_2]_2^0]_1^0]_2^0]_0^0.$$

A remark: if *Alice* (or *Oscar*) applies again $\alpha_{Alice}$ to the P system $\Pi'$, then the behaviour fails on the membrane structure $[_0 [_2 [_1 \alpha_1 [_2 x_2, m_{Bob}]_2^0]_1^-]_2^-]_0^-$.

In the same manner, if the object $\alpha_{Bob}$ is directly applied in the $CMS\ \Pi$, the failing appears in the membrane structure

$$[_0 [_1 x_1 [_2 x_2, m_{Bob}]_2^0]_1^0 [_2 [_1 \beta_1 [_3 x_3, ID_{Bob}]_3^0]_1^+]_2^+]_0^+.$$

*Remark 1.*

1. The development rules $(1), (3), (4), (5)$ from Example 2 can be considered general rules, used by any such a $CMS$. The particularity of a $CMS$ is represented by the development rules (2) and (6), which establish the paths in the tree and respectively modify the $CMS\ \Pi$ in $\Pi'$.
2. The manner of defining the development rules (4) in Example 2 represents a "peaceful" variant of using a $CMS$: the system is waiting until the legal objects $\alpha_{Alice}$ and respectively $\alpha_{Bob}$ are introduced.
   If these rules are replaced by
   $(4')$ $(a)$ $[_i \alpha x_i \rightarrow \beta]_i^-$ for all $\alpha \neq \alpha_i$,
   $(b)$ $[_i \alpha x_i \rightarrow \beta]_i^+$ for all $\alpha \neq \beta_i$,
   and $\beta \in V \setminus T$ is an object which destroys the whole content of $\Pi$ or $\Pi'$, we will obtain a variant where any illegal attempt leads to the erasing of all informations in the $CMS\ \Pi$ (respectively $\Pi'$).
   Actually, a hybrid variant is recommended, where the rules $(4a)$ are preserved as in Example 2, but the development rules $(4b)$ are replaced by $(4'b)$.

If the cryptanalist *Oscar* intercepts the $CMS\ \Pi$:

1. Without any additional information, the access to the message $m_{Bob}$ is impossible for him.
2. If he knows $\alpha_{Alice}$, then he can find $ID_{Bob}$. But any attempt to find the message $m_{Bob}$ without using $\alpha_{Bob}$ can lead to the destruction of this message (when the object $\beta$ and its development rules are defined in the system $\Pi'$).
3. If *Oscar* knows $\alpha_{Bob}$, applying it to $\Pi$ has no effect.
   Of course, if *Oscar* has both identification enzymes, his attack will be successful.

As a supplementary precaution, $ID_{Bob}$ can generate in the skin – before it goes out – another object $\gamma \in V \setminus T$. This object will ensure a protection of the new P system $\Pi'$, by eliminating at the first step any object different from $\alpha_{Bob}$, which is trying to find a path for the message $m_{Bob}$.

## 3 Authentication of Messages with Confirmation

In the protocol constructed above, if *Oscar* knows $\alpha_{Alice}$, then he has a possible succesfull attack of the following type:

1. He applies $\alpha_{Alice}$ in the $CMS\ \Pi$ and finds $ID_{Bob}$.
2. Then, he constructs a new $CMS\ \Pi_1$ with $ID_{Bob}$ and his own message $m_{Oscar}$; this $CMS$ is submitted to *Alice*.

In this way, *Alice* will accept the message $m_{Oscar}$ as being correct and she believes that *Bob* is the sender.

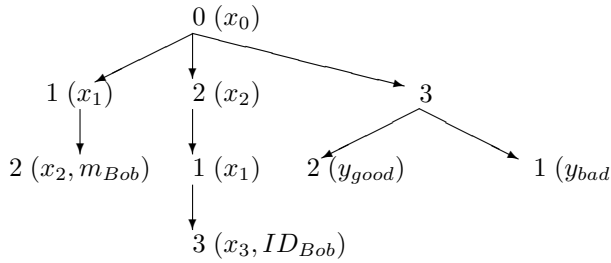
To avoid this attack, another message authentication protocol, with confirmation, will be proposed. This protocol extends the variant we presented above, with a password submitted by *Bob* when *Alice* asks him to certify.

Let us suppose that *Alice* receives a $CMS\ \Pi$ and wishes to find the message $m_{Bob}$, authenticated by *Bob*. She follows the next protocol:

1. *Alice* introduces $\alpha_{Alice}$ in the $CMS$ $\Pi$ and obtains $ID_{Bob}$; in the same time $\Pi$ is transformed in another P system $\Pi'$.
2. *Alice* finds the sender and submits an authentication request to *Bob*, using his address contained in $ID_{Bob}$.
3. *Bob* submits the password $\gamma$;
4. *Alice* introduces the object $\alpha_{Bob}\gamma$ in $\Pi'$ and obtains the message $m_{Bob}$.

In fact, $\gamma$ acts as a supplementary checking: it allows or forbids the message $m_{Bob}$ to go out of the skin. Usually, the introduction of $\alpha_{Bob}$ only has the same effect as the introduction of an illegal object: the loss of the message.

*Example 3.* Let us consider a $CMS$ having a similar membrane structure as that from Example 2, where a new membrane is added in the skin:




Here $y_{good}, y_{bad} \in V \setminus T$ are new objects. The development rules are the same as in Example 2, where only the rule $(5b)$ is restricted to $1 \leq i \leq 3$.

Initially, all polarities are neutral.

At the request of *Alice*, *Bob* can use one of two passwords: $\gamma_{good}$, which confirms the message and allows its going out of the skin, and $\gamma_{bad}$, which blocks the message going out of the skin (this object can be used by *Bob* in various situations: he is not sure of the identity of *Alice*, he changes his mind and wants to stop the transmission of the message $m_{Bob}$, etc).

Besides the rules $(1) - (6)$ defined above (with $(5b)$ restricted to $1 \leq i \leq 3$) the following development rules are added:

7. (a) $\gamma_{good}[_0]_0^x \to [_0\gamma_{good}]_0^x, \quad \gamma_{good}[_i]_i^0 \to [_i\gamma_{good}]_i^-, \quad i = 2, 3;$
   (b) $\gamma_{bad}[_0]_0^x \to [_0\gamma_{bad}]_0^x, \quad \gamma_{bad}[_i]_i^0 \to [_i\gamma_{bad}]_i^-, \quad i = 1, 3;$
   (the passwords $\gamma_s$ reach the membrans where there are the objects $y_s$ $s \in \{good, bad\}$; the paths are marked with a negative polarity.
8. $[_i y_s]_i^- \to y_s, \quad s \in \{good, bad\}, \ 1 \leq i \leq 3;$
   (the object $y$ is transported in the skin membrane, that labelled by 0);
9. (a) $[_0 y_{good}]_0^+ \to [_0]_0^0,$
   (b) $y_{bad}[_2]_2^+ \to [_2]_2^-$
   ($y_{good}$ neutralises the skin polarity; $y_{bad}$ changes the polarity of a membrane from the path of the message, by blocking its output).
10. $[_0 m_{Bob}]_0^0 \to m_{Bob}[_0]_0^0$
    (the message can go out only if the skin is neutrally polarized).

In this example we assume that all actions are syncronized: the time of entering membranes is the same for all objects and all membranes. Thus, the membrane labelled by 2 can be blocked before the message $m_{Bob}$ reaches this node.

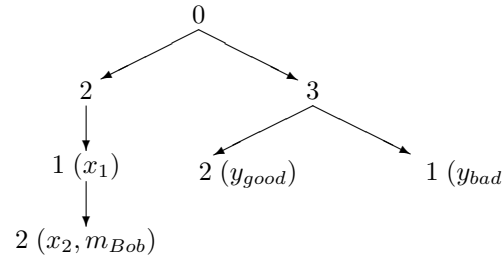
Let us see how this $CMS$ works: *Alice* receives the following configuration:

$$[_0[_1x_1[_2x_2, m_{Bob}]_2^{0}]_1^{0}[_2x_2[_1x_1[_3x_3, ID_{Bob}]_3^{0}]_1^{0}]_2^{0}[_3[_2y_{good}]_2^{0}[_1y_{bad}]_1^{0}]_3^{0}]_0^{0}.$$

After $\alpha_{Alice}$ is introduced in the skin, by transformations similar to those in Example 2, the system becomes

$$ID_{Bob}[_0[_2[_1x_1[_2x_2, m_{Bob}]_2^{0}]_1^{0}]_2^{+}[_3[_2y_{good}]_2^{0}[_1y_{bad}]_1^{0}]_3^{0}]_0^{0},$$

represented by the tree




At this moment *Alice* finds that the $CMS$ was submitted by *Bob*. From $ID_{Bob}$ she extracts the coordinates of *Bob* and she requires him to confirm the message. Two situations arise now:

1. *Bob* agrees and submits the password $\gamma_{good}$. *Alice* introduces in the skin of the system the objects $\alpha_{Bob}$ and $\gamma_{good}$; the membrane structure passes through the following configurations:
   $\alpha_{Bob}\gamma_{good}[_0[_2[_1x_1[_2x_2, m_{Bob}]_2^{0}]_1^{0}]_2^{+}[_3[_2y_{good}]_2^{0}[_1y_{bad}]_1^{0}]_3^{0}]_0^{0}$,
   $[_0\beta_0\gamma_{good}[_2[_1x_1[_2x_2, m_{Bob}]_2^{0}]_1^{0}]_2^{+}[_3[_2y_{good}]_2^{0}[_1y_{bad}]_1^{0}]_3^{0}]_0^{+}$,
   $[_0[_2\beta_2[_1x_1[_2x_2, m_{Bob}]_2^{0}]_1^{0}]_2^{+}[_3\gamma_{good}[_2y_{good}]_2^{0}[_1y_{bad}]_1^{0}]_3^{-}]_0^{+}$,
   $[_0[_2[_1\beta_1[_2x_2, m_{Bob}]_2^{0}]_1^{+}]_2^{+}[_3[_2\gamma_{good}y_{good}]_2^{-}[_1y_{bad}]_1^{0}]_3^{-}]_0^{+}$,
   $[_0[_2[_1[_2\beta_2, m_{Bob}]_2^{+}]_1^{+}]_2^{+}[_3\gamma_{good}y_{good}[_1y_{bad}]_1^{0}]_3^{-}]_0^{+}$,
   $[_0[_2[_1m_{Bob}[_2\beta_2]_2^{0}]_1^{+}]_2^{+}\gamma_{good}y_{good}[_1y_{bad}]_1^{0}]_0^{+}$,
   $[_0[_2m_{Bob}[_1[_2\beta_2]_2^{0}]_1^{0}]_2^{+}\gamma_{good}[_1y_{bad}]_1^{0}]_0^{+}$,
   $[_0m_{Bob}[_2[_1[_2\beta_2]_2^{0}]_1^{0}]_2^{+}\gamma_{good}[_1y_{bad}]_1^{0}]_0^{+}$,
   $m_{Bob}[_0[_2[_1[_2\beta_2]_2^{0}]_1^{0}]_2^{+}\gamma_{good}[_1y_{bad}]_1^{0}]_0^{+}$,
   and the message $m_{Bob}$ becomes accessible to *Alice*.
2. *Bob* submits the password $\gamma_{bad}$. In this case, the steps are the following:
   $\alpha_{Bob}\gamma_{bad}[_0[_2[_1x_1[_2x_2, m_{Bob}]_2^{0}]_1^{0}]_2^{+}[_3[_2y_{good}]_2^{0}[_1y_{bad}]_1^{0}]_3^{0}]_0^{0}$,
   $[_0\beta_0\gamma_{bad}[_2[_1x_1[_2x_2, m_{Bob}]_2^{0}]_1^{0}]_2^{+}[_3[_2y_{good}]_2^{0}[_1y_{bad}]_1^{0}]_3^{0}]_0^{+}$,
   $[_0[_2\beta_0[_1x_1[_2x_2, m_{Bob}]_2^{0}]_1^{0}]_2^{+}[_3\gamma_{bad}[_2y_{good}]_2^{0}[_1y_{bad}]_1^{0}]_3^{-}]_0^{+}$,
   $[_0[_2[_1\beta_1[_2x_2, m_{Bob}]_2^{0}]_1^{+}]_2^{+}[_3[_2y_{good}]_2^{0}[_1\gamma_{bad}y_{bad}]_1^{-}]_3^{-}]_0^{+}$,
   $[_0[_2[_1[_2\beta_2, m_{Bob}]_2^{+}]_1^{+}]_2^{+}[_3[_2y_{good}]_2^{0}\gamma_{bad}y_{bad}]_3^{-}]_0^{+}$,
   $[_0[_2[_1m_{Bob}[_2\beta_2]_2^{0}]_1^{+}]_2^{+}[_2y_{good}]_2^{0}\gamma_{bad}y_{bad}]_0^{+}$,
   $[_0[_2m_{Bob}[_1[_2\beta_2]_2^{0}]_1^{0}]_2^{-}[_2y_{good}]_2^{0}\gamma_{bad}]_0^{+}$,
   and the process fails, because $m_{Bob}$ cannot go out through membrane 2 which was negatively polarized by $y_{bad}$ (using the rule $(9b)$).

We can remark that if in the second case *Alice* applies only $\alpha_{Bob}$ (without the password $\gamma$), then the message $m_{Bob}$ cannot pass through the skin, because the skin is positively polarized by the rule $(1b)$, and the neutralisation of this membrane can be performed only by $y_{good}$.

## 4 Final Remarks

This paper presents two variants of message authentication protocols, using the framework of P systems. The approach is mainly intended to prove the possible usefulness of P systems in addressing this cryptographic issue, to test the versatility of membrane computing area/language, rather than being a practical proposal. The constructions are very easy, while their security (not examined here from a computational complexity point of view) depends on the unicity of objects which can interact, and on the blocking possibility of the P systems in case of wrong action.

Among other features which can be considered, we mention:

1. At the second or the third attempt to activate the $CMS$ with wrong "enzymes", the system is self-destroyed. Such a construction is easy to be carried.
2. If the system failed, then the application of the illegal object transforms the initial CMS in another one, $\Pi''$; for this, another pair of identification enzymes $(\alpha''_{Alice}, \alpha''_{Bob})$ are necessary.
3. In the same CMS structure several messages and several identifications data $ID$ can be introduced; the basic idea of CMSs will not be essentially changed.
4. The CMS structure can be adapted for only one use (for example, in the case of bills).
5. A time interval of validity can be assigned to each CMS; after the expiration of this time, the object $\alpha_{Alice}$ will be not recognised as legal by the CMS.

The authentication type (and thus the signature type) we present here is more similar to ologaf signatures than to digital signatures. Here the CMS $\Pi$ does not depend on the message $m_{Bob}$, but only on the structure CMS chosen by *Bob*.

We intend to continue the study of these ideas by looking for algorithms of signatures and message authentication where the complexity will be also considered. In this moment this goal seems to be quite difficult, because we know that NP-complete problems are easy to be solved in the P systems area (e.g., when membrane division is available).

## References

1. A. Atanasiu, P systems and arithmetic calculus, *Report 14/00* of the Research Group on Mathematical Linguistics, Universitat Rovira i Virgili Tarragona, Spain, 2000.

2. A. Atanasiu, Arithmetic with membranes, *Romanian J. of Information Science and Technology*, 4, 1-2 (2001), 5–20.
3. A. Atanasiu, C. Martin-Vide, Recursive calculus with membranes, *Fundamenta Informaticae*, 49, 1-3 (2002), 45–59.
4. A. Atanasiu, C. Martin-Vide, P systems and context-free languages, in *Actas del Primer Congreso Espanol de Algoritmos Evolutivos y Bioinspirados (AEB'02)* (E. Alba, F. Fernandez, J.A. Gomez, F. Herrera, J.I. Hidalgo, J. Lanchares, J.J. Merelo et J.M. Sanchez, Eds.), Centro Universitario de Merida, Merida, 2002, 341–346.
5. S.N. Krishna, R. Rama, A variant of P systems with active membranes: Solving NP-complete problems, *Romanian J. of Information Science and Technology*, 2, 4 (1999), 357–367.
6. R.C. Merkle, A digital signature based on a conventional encryption function, *Proc. of Advances in Cryptology - CRYPTO '87*, Springer-Verlag, 1990, 369–378.
7. Gh. Păun, Computing with membranes - A variant: P systems with polarized membranes, *Intern. J. of Foundations of Computer Science*, 11, 1 (2000), 167–182.
8. Gh. Păun, Computing with membranes (P systems); Attacking NP-complete problems, *Unconventional Models of Computing* (I. Antoniou, C.S. Calude, M.J. Dinnen, Eds.), Springer-Verlag, 2000, 94–115.
9. Gh. Păun, G. Rozenberg, A guide to membrane computing, *Theoretical Computer Science*, 287, 1 (2002), 73–100.
10. G. Rozenberg, A. Salomaa, Eds., *Handbook of Formal Languages*, Springer-Verlag, 1997.
11. B. Schneier, *Applied Cryptography*, Second Edition, John Wiley and Sons, 1996.
12. D. Stinton, *Cryptographie, Theorie et Pratique*, Intern. Thomson Publ. France, 1996.

# Eilenberg P Systems

Tudor Bălănescu[1], Marian Gheorghe[2], Mike Holcombe[2], and Florentin Ipate[1]

[1] Faculty of Sciences, Piteşti University
Str. Targu din Vale 1, 0300 Piteşti, Romania
f.ipate@ifsoft.ro
[2] Department of Computer Science, Sheffield University
Regent Court, Portobello Street, Sheffield, S1 4DP, UK
{m.gheorghe,m.holcombe}@dcs.shef.ac.uk

**Abstract.** A class of P systems, called EP systems, with string objects processed by evolution rules distributed alongside the transitions of an Eilenberg machine, is introduced. A parallel variant of EP systems, called EPP systems, is also defined and the power of both EP and EPP systems is investigated in relationship with three parameters: number of membranes, states and set of distributed rules. It is shown that EPP systems represent a promising framework for solving NP-complete problems. In particular linear time solutions are provided for the SAT problem.

## 1 Introduction

P systems were introduced in the history making paper [21] by Gh. Păun. One of the main classes of P systems is that of string objects where the evolution rules are defined as string rewriting operations (see [22,21,6,18,19,20]) (an up-to-date bibliography of the whole area may be found at the web address http://psystems.disco.unimib.it). Because rewriting alone even in the context of a highly parallel environment of a membrane structure is not enough to lead to characterizations of recursively enumerable languages, various other features have been considered, such as a *priority relationship* over the set of rules, *permitting* or *forbidding* conditions associated with rules, *restrictions on the derivation mode*, the possibility to control the *membrane permeability* [6] etc (for more details see [22]). In general the most used priority relationship on the set of rewriting rules is a partial order relationship, well studied in the context of generative mechanisms with restrictions in derivation [4]. In this paper the priority relationship will be replaced by a transition diagram associated with an Eilenberg machine giving birth to two classes of Eilenberg systems, a sequential version and a parallel one, called *EP systems* and *EPP systems,* respectively. In both variants, each transition has a specific set of evolution rules acting upon the string objects contained in different regions of the membrane system. The system will start in a given state and with an initial set of string objects. Given a state and a current set of string objects, in the case of EP systems, the machine will evolve by applying rules associated with one of the transitions going out from the current state. The system will resume from the destination state of the

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 43–57, 2003.

current transition. In the parallel variant, instead of one state and a single set of string objects we may have a number of states, called *active states,* that are able to trigger outgoing transitions and such that each state hosts a different set of string objects; all the transitions emerging from every active state may be triggered once the rules associated with them may be applied; then the system will resume from the next states, which then become active states. EP systems are models of cells evolving under various conditions when certain factors may inhibit some evolution rules or some catalysts may activate other rules. EPP systems introduce a parallel behaviour of the system in respect of the transitions emerging from active states, model cellular division and parallel development of the new born cells as well as cell collision when multiple transitions join a target state. The EP model has some similarities with the grammar systems controlled by graphs [3], replacing a one-level structure, which is the current sentential form, with a hierarchical structure defined by a membrane element. It is also close to the state based model defined by Ji [15] in relation to the living cell, called molecular machine. Whereas Ji's molecular machine is a sort of Mealy machine [17], the proposed model penetrates inside of the cell as usually P systems do. On the other hand, this P system variant may be viewed as an Eilenberg machine [5] having sets of evolution rules as basic processing relationships. EP systems are also similar to the Eilenberg machines based on distributed grammar systems [8]. Eilenberg machines, generally known under the name of X machines [5], have been initially used as a software specification language [10], further on intensively studied in connection with software testing [13], but also utilized as a model of methabolic pathways [11], bee colony behaviour [9], or reactive agents [16]; a survey of the whole area at the end of 2000 is given by [12]. EPP systems present similarities with Petri nets [14], but also with communicating X-machine systems (see [1], [16]). The main difference between EPP systems and the other models consists in a structured framework which is a cell like structure, where at every step, each string object in each region of the system may be transformed by applying some evolution rules. The style of triggering transitions in parallel recalls replicated rewriting derivation mode studied for some classes of P systems [19].

In this paper it is investigated the power of both systems in connection with three parameters: number of membranes, states and set of distributed rules. On the other hand it is shown that EPP systems represent a promising framework for solving NP-complete problems; in particular linear time solutions are provided for SAT problem. The last result relies heavily on similarities between EPP systems and P systems with replicated rewriting [19], showing that more connections with these types of P systems might be further investigated.

## 2 Definitions

**Definition 1.** *A* stream Eilenberg machine *is a tuple*

$$X = (\Sigma, \Gamma, Q, M, \Phi, F, I, T, m_0),$$

*where:*

- $\Sigma$ *and* $\Gamma$ *are finite sets called the* input *and the* output *alphabets, respectively;*
- $Q$ *is the finite set of* states*;*
- $M$ *is a (possibly infinite) set of* memory symbols*;*
- $\Phi$ *is a set of basic partial relations on* $\Sigma \times M \times M \times \Gamma^*$;
- $F$ *is the* next state function $F : Q \times \Phi \to 2^Q$;
- $I$ *and* $T$ *are the sets of* initial *and* final *states;*
- $m_0$ *is the* initial memory *value.*

**Definition 2.** *An* EP system *is a construct* $E\Pi = (\mu, X)$, *where* $\mu$ *is a membrane structure consisting of* $m$ *membranes, with the membranes and the regions labelled in a one to one manner with the elements* $1, \ldots, m$ *and an Eilenberg machine whose memory is defined by the regions* $1, \ldots, m$ *of* $\mu$. *The Eilenberg machine is a system*

$$X = (V, \Gamma, Q, M_1, \ldots, M_m, \Phi, F, I),$$

*having the following properties*

- $V$ *is the* alphabet *of the system;*
- $\Gamma, Q, F$ *are as in Definition 1;* $\Gamma \subseteq V$, *is called now* terminal alphabet*;*
- $M_1, \ldots, M_m$ *are finite languages over* $V$ *and represent the initial values occurring in the regions* $1, \ldots, m$ *of the system;*
- $\Phi = \{\Phi_1, \ldots, \Phi_p\}$, $\Phi_i = (R_{i,1}, \ldots R_{i,m})$, $1 \leq i \leq p$ *and* $R_{i,j}$ *is a set of evolution rules (possibly empty) associated with region* $j$, *of the form* $X \to (u, tar)$, *with* $X \in V$, $u \in V^*$, $tar \in \{here, out, in\}$; *the indication here will be omitted and the rule will be written* $X \to u$;
- $I = \{q_0\}$, $q_0 \in Q$ *is the initial state; all the states are final states (equivalent to* $Q = T$*).*

It may be observed that the set $\Sigma$ and $m_0$ from Definition 1 are no longer used in the context of EP systems. In fact, these concepts have been replaced by $V$ and $M_1, \ldots, M_m$, respectively.

A P system has $m$ sets of evolution rules, each one associated with a region. An EP system has the evolution rules distributed among $p$ components $\Phi_i$, $1 \leq i \leq p$, each one containing $m$ sets of evolution rules.

A computation in $E\Pi$ is defined as follows: it starts from the initial state $q_0$ and an initial configuration of the memory defined by $M_1, \ldots M_m$ and proceeds iteratively by applying in parallel rules in all regions, processing in each one all strings that can be rewritten; in a given state $q$, each string is processed by a single rule following the target indication of that rule (for instance, when rewriting $xXv$ by a rule $X \to (u, tar)$, the string $xuv$ obtained will be send to the region indicated by $tar$, with the usual meaning in P systems (see [2], [22], [6])); if several rules may be applied to a string, then one rule and one symbol to which it is applied are randomly chosen; the rules are from a component $\Phi_i$ which is associated with one of the transitions emerging from the current state $q$ and

the resulting strings constitute the new configuration of the membrane structure with the associated regions; the next state, belonging to $F(q, \Phi_i)$, will be the target state of the selected transition. The result (a set of strings containing only symbols from $\Gamma$) is collected outside of the system at the end of a halting computation.

EPP systems have the same underlying construct $(\mu, X)$, with the only difference that instead of one single membrane structure, it deals with a set of instances having the same organization $(\mu)$, but being distributed across the system. More precisely, these instances are associated with states called *active states;* these instances can divide up giving birth to more instances or collide into single elements depending on the current configuration of the active states and the general topology of the underlying machine. Initially only $q_0$ is an active state and the membrane configuration associated with $q_0$ is $M_1, \ldots, M_m$. All active states are processed in parallel in one step: all emerging transitions from these states are processed in parallel (and every single transition processes in parallel each string object in each region, if evolution rules match them).

*Cell division:* if $q_j$ is one of the active states, $M_{j,1}, \ldots, M_{j,m}$ is its associated membrane configuration instance, and $\Phi_{j,1}, \ldots, \Phi_{j,t}$ are $\Phi'$s components associated with the emerging transitions from $q_j$, then the rules occurring in $\Phi_{j,i}$, $1 \leq i \leq t$, are applied to the string objects from $M_{j,1}, \ldots, M_{j,m}$, the control passes onto $q_{j,1}, \ldots, q_{j,t}$, which are the target states of the transitions earlier nominated, with $M_{j,1,1}, \ldots, M_{j,m,1}, \ldots, M_{j,1,t}, \ldots, M_{j,m,t}$, their associated membrane configuration instances, obtained from $M_{j,1}, \ldots, M_{j,m}$, by applying rules of $\Phi_{j,1}, \ldots, \Phi_{j,t}$; the target states become active states, $q$ is desactivated and $M_{j,1}, \ldots, M_{j,m}$ vanish. Only $\Phi_{j,i}$ components that have rules matching the string objects of $M_{j,1}, \ldots, M_{j,m}$, are triggered and consequently only their target states become active and associated with memory instances $M_{j,1,i}, \ldots, M_{j,m,i}$. If none of $\Phi_{j,i}$ is triggered then in the next step $q$ is desactivated and $M_{j,1}, \ldots, M_{j,m}$ vanish too. If some of $\Phi_{j,i}$ are indicating the same component of $\Phi$ then the corresponding memory configurations $M_{j,1,i}, \ldots, M_{j,m,i}$ are the same as well; this means that always identical transitions emerging from a state yield the same result.

*Cell collision:* if $\Phi_1, \ldots, \Phi_t$ enter the same state $r$ and some or all of them emerge from active states, then the result associated with $r$ is the union of membrane instances produced by those $\Phi'_i$s emerging from active states and matching string objects from their membrane instances.

A computation of an EP (EPP) system halts when none of the rules associated with the transitions emerging from the current states (active states) may be applied.

The language computed by a system $E\Pi$ is denoted by $L(E\Pi)$ and consists of all strings over $\Gamma$ that are sent out of the system during a halting computation.

The family of languages generated by EP (EPP) systems with at most $m$ membranes, at most $s$ states and using at most $p$ sets of rules is denoted by $EP_{m,s,p}(EPP_{m,s,p})$. If one of these parameters is not bounded, then the corresponding subscript is replaced by $*$. An EP (EPP) system is called pure when

$V = \Gamma$ and the family of languages generated by such systems with the above considered parameters is denoted by $PEP_{m,s,p}$ ($PEPP_{m,s,p}$).

A matrix grammar with appearance checking in the *binary normal form* (for more details see [3]) is a construct $G = (N, T, S, M, F)$ where $N = N_1 \cup N_2 \cup \{S, \#\}$, with these three sets mutually disjoint, and the matrices in $M$ with one of the following forms:

- 1. $(S \to ZB)$, with $Z \in N_1$, $B \in N_2$,
- 2. $(X \to Y, A \to x)$, with $X, Y \in N_1$, $A \in N_2$, $x \in (N_2 \cup T)^*$,
- 3. $(X \to Y, A \to \#)$, with $X, Y \in N_1$, $A \in N_2$,
- 4. $(X \to \lambda, A \to x)$, with $X \in N_1$, $A \in N_2$, $x \in T^*$.

The set $F$ consists only of rules $A \to \#$ appearing in matrices of type 3. The family of languages produced by these devices is denoted by $MAT_{ac}$. When rules of type 3 are not used, then the corresponding family of languages is denoted by $MAT$. With $RE$ denoting the set of recursively enumerable languages, the following relations hold $MAT \subset MAT_{ac} = RE$, where the inclusion is proper. It is also known that two nonterminals used in rules $A \to \#$ suffice to generate all $RE$ languages [7].

## 3 Computational Power of EP and EPP Systems

It has been noted that rewriting alone even in a highly structured and parallel environment of a membrane, does not suffice for characterizing the set of recursively enumerable languages [6]. Thus various priority relationships have been considered. In this paper, instead of a partial order relationship on the set of rewriting rules, the rules are first distributed among sets $\Phi_i$ and then controlled by the next state function which selects the set of rules to be applied in the current state. Let us first consider an example to illustrate how an EP system works. Let $E\Pi = ([_1]_1, X)$, where $X$ contains the following elements:

- $V = \{A, A', B, B', \#, a, b, c\}$;
- $\Gamma = \{a, b, c\}$;
- $\Phi = \{\Phi_1, \Phi_2, \Phi_3, \Phi_4, \Phi_5\}$, where
  - $\Phi_1 = (\{A \to aAb, A' \to B', B' \to \#\})$,
  - $\Phi_2 = (\{B \to Bc, B' \to A', A' \to \#\})$,
  - $\Phi_3 = (\{A \to ab, A' \to B', B' \to \#\})$,
  - $\Phi_4 = (\{B \to (c, out), B' \to \lambda, A' \to \#\})$,
  - $\Phi_5 = \{\# \to \#\}$;
- $Q = \{1\}$, $I = \{1\}$;
- $F(\Phi_i, 1) = \{1\}, 1 \leq i \leq 5$;
- $M_1 = \{aAbBc, A'\}$.

The region 1 always has one of the following forms:

1. $\{a^n Ab^n Bc^{n+k}, A'\}, n \geq 1, k \geq 0$,
2. $\{a^{n+1} Ab^{n+1} Bc^{n+k}, B'\}, n \geq 1, k \geq 0$,

3. $\{a^{n+1}b^{n+1}Bc^{n+k}, B'\}, n \geq 1, k \geq 0$,
4. $\{a^{n+1}b^{n+1}Bc^{n+1+k}, A'\}, n \geq 1, k \geq 0$,
5. $\{y, \#\}, y \in V^*$,
6. $\{\lambda\}$.

This may be proved by induction on the number of the computation steps. Initially it has the form 1, with $n = 1, k = 0$. Next, it may be seen that:

- from 1 it results 5 (applying $\Phi_2$ or $\Phi_4$), or 2 (applying $\Phi_1$), or 3 (applying $\Phi_3$);
- from 2 it results 5 (applying $\Phi_1$ or $\Phi_3$), or 1 (applying $\Phi_2$), or 6 (applying $\Phi_4$);
- from 3 it results 5 (applying $\Phi_1$ or $\Phi_3$), or 4 (applying $\Phi_2$), or 6 (applying $\Phi_4$);
- from 4 it results 5 (applying $\Phi_2$ or $\Phi_4$), or 3 (applying $\Phi_3$);
- from 5 it results 5 (applying any set that matches $y$; $\Phi_5$ may be always applied);
- in the form 6 the computation halts.

The only way to obtain terminal strings outside the system is to apply $\Phi_4$ on the form 3 and to send out a string of the form $a^{n+1}b^{n+1}c^{n+1+k}, n \geq 1, k \geq 0$. The language of all strings of this form is not context-free. Some lessons may be learned from this simple example: the computation process of a set of words cannot be split down into independent computations of the individual elements (this property will be used further on in some more general proofs); using only one membrane and one state (i.e., no control mechanism imposed by the next state function $F$), non-context-free languages may be generated.

If we consider the above specification as an EPP system denoted by $E\Pi\Pi$ then $L(E\Pi\Pi) = \emptyset$, because all first four components are triggered, # is introduced, and the computation never halts. The following $E\Pi\Pi'$ using the above defined $V$ and $\Gamma$, but redefining $\Phi, Q, F$ and $M_1$

- $\Phi = \{\Phi_1, \Phi_2, \Phi_3, \Phi_4, \Phi_5\}$, where
  - $\Phi_1 = (\{A \to aAb, A \to aA'b\})$,
  - $\Phi_2 = (\{B \to Bc, B \to B'c\})$,
  - $\Phi_3 = (\{A' \to ab\})$,
  - $\Phi_4 = (\{B' \to B'c, B' \to Bc\})$,
  - $\Phi_5 = \{B \to (c, out)\}$;
- $Q = \{1, 2, 3, 4\}$, $I = \{1\}$;
- $F(\Phi_1, 1) = \{2\}$,
  $F(\Phi_2, 2) = \{1\}$,
  $F(\Phi_3, 1) = \{3\}$,
  $F(\Phi_4, 3) = \{3\}$,
  $F(\Phi_5, 3) = \{4\}$;
- $M_1 = \{AB\}$;

leads to $L(E\Pi\Pi') = L(E\Pi)$. The underlying system computes the same language when it is to be considered an EP system as well.

Some preliminary results follow directly from definitions ([$P$] means that the involved relationships hold for both $P$ being present in both its members or none of them).

**Lemma 1.** *(i)* $PEP_{m,s,p} \subseteq EP_{m,s,p}$, $m, s, p \geq 1$;
*(ii)* $[P]EP_{m,s,p} \subseteq [P]EP_{m+k_1,s+k_2,p+k_3}$, $k_1, k_2, k_3 \geq 0$.

The next result is similar to Lemma 2 in [6], giving also details about the number of states and rules used by the proof.

**Lemma 2.** $EP_{m,s,p} \subseteq PEP_{m+1,s+1,p+2}$, $m, s, p \geq 1$.

*Proof.* Let us consider an EP system $E\Pi = (\mu, X)$, with $\mu$ a membrane structure with $m$ regions and $X$ a machine given by

$$X = (V, \Gamma, Q, M_1, \ldots M_m, \Phi, F, I),$$

according to Definition 2. We construct now the pure EP system $E\Pi' = (\mu', X')$, where $\mu' = [_0\mu]_0$, and

$$X' = (V', V', Q', M'_0, M'_1, \ldots M'_m, \Phi', F', I),$$

with

- $V' = V \cup \{f, \}, f \notin V$;
- $Q' = Q \cup \{q_0\}$, $q_0 \notin Q$;
- $M'_i = \{fw \mid w \in M_i\}$, $1 \leq i \leq m$; $M'_0 = \emptyset$;
- $\Phi' = \{\Phi'_i \mid \Phi_i \in \Phi, 1 \leq i \leq p\} \cup \{\Phi'_{p+1}, \Phi'_{p+2}\}$, where
  $\Phi'_i = (\emptyset, R_{i,1}, \ldots R_{i,m})$, for $\Phi_i = (R_{i,1}, \ldots R_{i,m})$,
  $\Phi'_{p+1} = (\{a \to (a, in) \mid a \in V \setminus \Gamma\}, \emptyset, \ldots \emptyset)$ and
  $\Phi'_{p+2} = (\{f \to (\lambda, out)\}, \emptyset, \ldots \emptyset)$;
- $F'(q, \Phi'_i) = F(q, \Phi_i)$, $q \in Q$, $\Phi_i \in \Phi$, $1 \leq i \leq p$, and
  $F'(q, \Phi'_{p+1}) = \{q_0\}$, $F'(q_0, \Phi'_{p+2}) = \{q_0\}$.

According to the above construction any string $x$ processed by $E\Pi$ exits the system iff $fx$ arrives in region 0 of $E\Pi'$, in a state $q \in Q$. Indeed any application of a set of rules $\Phi_i$ in $E\Pi$ is simulated in $E\Pi'$ by $\Phi'_i$. Any string in region 0 is checked to contain only elements from $\Gamma$, by applying $\Phi'_{p+1}$. All strings in region 0 which contain some symbols in $V \setminus \Gamma$ fall down into region 1. Otherwise the rule $f \to (\lambda, out)$ occurring in $\Phi'_{p+2}$ pops out strings containing only symbols from $\Gamma$. Hence $L(E\Pi) = L(E\Pi')$. □

**Theorem 1.** $MAT = EP_{4,1,1} \subseteq PEP_{5,2,3}$.

*Proof.* An EP system with $m$ membranes, a single state and only one set of rules is an extended rewriting system [6], and for these systems it has been shown that 4 membranes suffice to generate $MAT$, and consequently the result holds. □

It is also known that a graph controlling the derivation, = 1 style, in grammar systems, without appearance checking feature, produces exactly $MAT$ (Theorem 4.7 in [3]). What about an EP system with only one membrane? The only difference between such a system and a grammar system as described above is that in an EP system all strings are kept in one membrane and are rewritten in parallel in one step. From Theorem 1 we have learned that without spreading the rules and using only one state we do not get more than $MAT$ languages.

**Theorem 2.** $EP_{1,1,*} = PEP_{2,2,*} = RE$.

*Proof.* According to Turing-Church thesis and Lemma 2, it is only to prove the inclusion $RE \subseteq EP_{1,1,*}$. As usually in such cases a matrix grammar with appearance checking in the binary normal form $G = (N_1 \cup N_2 \cup \{S, \#\}, T, S, M, F)$ is considered. It is also assumed that the rules are labelled in a one to one manner with $m_1, \ldots, m_{k_1}$ (the matrices of type 2), $m_{k_1+1}, \ldots, m_{k_2}$ (the matrices of type 3), and $m_{k_2+1}, \ldots, m_{k_3}$ (the matrices of type 4). The following EP system is constructed $E\Pi = (\mu, X)$, where $\mu$ is a membrane structure consisting of a single membrane, and $X$ an Eilenberg machine

$$X = (N_1 \cup N_2 \cup \{S, \#\} \cup T, T, \{q_0\}, M_1, \Phi, F, \{q_0\}),$$

with

- $M_1$ containing $Z$ and $B$, the symbols occurring in the right hand side of the matrix of type 1;
- $F(q_0, \Phi_i) = \{q_0\}$, for any $\Phi_i \in \Phi$;
- the set $\Phi$ containing
  - for each matrix $m_i = (X \to Y, A \to x)$, $1 \leq i \leq k_1$, of type 2, a set of rules
    $\Phi_i = (\{A \to x, X \to Y\} \cup \{U \to \# \mid U \in N_1 \cup N_2, U \neq X, U \neq A\})$;
  - for each matrix $m_i = (X \to Y, A \to \#)$, $k_1 + 1 \leq i \leq k_2$, of type 3, a set of rules
    $\Phi_i = (\{A \to \#, X \to Y\} \cup \{U \to \# \mid U \in N_1, U \neq X\})$;
  - for each matrix $m_i = (X \to \lambda, A \to x)$, $k_2 + 1 \leq i \leq k_3$, of type 4, a set of rules
    $\Phi_i = (\{A \to x, X \to (\lambda, out)\} \cup \{U \to \# \mid U \in N_1 \cup N_2, U \neq X, U \neq A\})$
  - a new rule is considered in the set $\Phi_0 = (\{\# \to \#\})$;

The computation will start with $\{Z, B\}$ in the main region defined by the skin (main) membrane. If $\{X, uAv\}$ is the current content of the region, then this corresponds to the sential form $XuAv$ associated with the grammar $G$. A matrix $m_i$, $1 \leq i \leq k_1$ is succesfully applied to $XuAv$ iff the corresponding rules of $\Phi_i$ are applied in one step in parallel to both $X$, and $uAv$. A failure in correctly applying this matrix leads to blocking the derivation process in $G$ and accordingly the introduction of # symbol in the current membrane. Similarly, the set of rules associated with matrices of types 3 and 4, simulate the use of these matrices in $G$. Any blocking derivation in $G$ is simulated in $E\Pi$ by introducing '#' symbol which leads to an endless computation (set $\Phi_0$) or to the situation when a terminal sequence is never sent out of the system. □

The proof of Theorem 2 says that only one membrane and one state suffice to compute all $RE$ languages, but with an unbounded number of rules. On the other hand the simulation is very efficient and natural, the number of steps involved in a computation in $E\Pi$ is exactly equal to the number of matrices required by the corresponding equivalent derivation of $G$. What can be said when all three parameters are bounded? The next result shows that by increasing either the number of membranes or the number of states, EP systems with a bounded number of functions that compute all $RE$ languages may be found.

**Theorem 3.** *(i)*$EP_{1,3,8} = RE$; *(ii)* $EP_{2,1,7} = RE$.

*Proof.* Again we consider a matrix grammar with appearance checking in the binary normal form $G = (N_1 \cup N_2 \cup \{S, \#\} \cup T, T, S, M, F)$. It is assumed that the rules are labelled in a one to one manner with $m_1, \ldots, m_{k_1}$ (matrices of type 2), $m_{k_1+1}, \ldots, m_{k_2}$ (matrices of type 3), and $m_{k_2+1}, \ldots, m_{k_3}$ (matrices of type 4). Also we assume that type 3 matrices (with appearance checking) utilize only two nonterminals $A_1$, $A_2$ [7]. We denote by $Dom_i$ the set of nonterminal symbols occurring in the left hand side of rules of matrices of type $i$, $1 \leq i \leq 4$.

(i) Let $E\Pi = (\mu, X)$ be an EP system with one membrane and three states, where $\mu$ is a membrane structure, and $X$ an Eilenberg machine. The underlying machine is given by

$$X = (N_1 \cup N_2 \cup \{S, \#, f\} \cup \{i_j \mid 0_j \leq i_j \leq (k_3)_j, 1 \leq j \leq 4\} \cup T, \\ T, Q, M_1, \Phi, F, \{q_1\}),$$

where $f$ and $i_j$ are new symbols and

- $Q = \{q_1, q_2, q_3\}$;
- $M_1$ contains $Zf$ and $Bf$, with $Z$ and $B$ the symbols occurring in the right hand side of the rule of type 1, and $f$ is the new symbol introduced by $X$;
- $\Phi = \{\Phi_1, \ldots, \Phi_8\}$, where
  - $\Phi_1 = (\{X \to Yi_1 \mid 1_1 \leq i_1 \leq (k_1)_1,$ and there is $i_1 : (X \to Y, A \to x)$ a matrix of type 2 $\}\cup$ $\{X \to \lambda i_3 \mid 1_3 \leq i_3 \leq (k_3)_3,$ and there is $i_3 : (X \to \lambda, A \to x)$ a matrix of type 4 $\}\cup$ $\{A \to xi_2 \mid 1_2 \leq i_2 \leq (k_1)_2,$ and there is $i_2 : (X \to Y, A \to x)$ a matrix of type 2 $\}\cup$ $\{A \to xi_4 \mid 1_4 \leq i_4 \leq (k_3)_4,$ and there is $i_4 : (X \to \lambda, A \to x)$ a matrix of type 4 $\}\cup$ $\{U \to \# \mid U \in N_1 \cup N_2, U \notin Dom_2 \cup Dom_4\})$,
  - $\Phi_2 = (\{i_j \to i_j - 1 \mid 1 \leq j \leq 4, 0_1 < i_1 \leq (k_1)_1,$ $0_2 < i_2 \leq (k_1)_2, 0_3 < i_3 \leq (k_3)_3,\ 0_4 < i_4 \leq (k_3)_4, \} \cup \{f \to \#\})$,
  - $\Phi_3 = (\{0_1 \to \lambda, 0_2 \to \lambda\} \cup \{i_j \to \# \mid i_j > 0, 1 \leq j \leq 4\})$,
  - $\Phi_4 = (\{f \to \lambda\})$,
  - $\Phi_5 = (\{0_3 \to \lambda, 0_4 \to (\lambda, out)\} \cup \{i_j \to \# \mid i_j > 0, 1 \leq j \leq 4\})$,
  - $\Phi_6 = (\{X \to Y, A_1 \to \# \mid (X \to Y, A_1 \to \#)$ matrix of type 3 $\}\cup$ $\{U \to \# \mid U \in N_1 \setminus Dom_3\})$,

  - $\Phi_7 = (\{X \to Y, A_2 \to \# \mid (X \to Y, A_2 \to \#)$ matrix of type 3 $\} \cup$ $\{U \to \# \mid U \in N_1 \setminus Dom_3\})$,
  - $\Phi_8 = (\{\# \to \#\})$;
- $F(q_1, \Phi_i) = \{q_1\}$, $i \in \{6, 7, 8\}$, $F(q_1, \Phi_1) = \{q_2\}$, $F(q_2, \Phi_2) = \{q_2\}$, $F(q_2, \Phi_3) = \{q_1\}$, $F(q_2, \Phi_4) = \{q_3\}$, $F(q_3, \Phi_5) = \{q_1\}$.

A computation in $E\Pi$ develops as follows:

- the process starts in state $q_1$ with initial configuration $\{Zf, Bf\}$;
- given a configuration $\{Xf, \alpha A\beta f\}$, one of $\Phi_1, \Phi_6$ or $\Phi_7$ may be applied; if $\Phi_1$ is selected then it meant to simulate either matrices $i : (X \to Y, A \to x)$ of type 2 or $i : (X \to \lambda, A \to x)$ of type 4, by applying $X \to Yi_1, A \to xi_2$ or $X \to \lambda i_3, A \to xi_4$ and leading to $\{Yi_1 f, \alpha x i_2 \beta f\}$ or $\{i_3 f, \alpha x i_4 \beta f\}$ in state $q_2$; in both cases rules of $\Phi_2$ are iteratively applied to check whether the previous rules came from the same matrix (either $i_1 = i_2$ or $i_3 = i_4$); if one index, $i_j$ reaches $0_j$ and the other not then a # symbol is introduced by $f \to \#$; if $\Phi_2$ is left before reaching $0_j$ then either $\Phi_3$ or $\Phi_5$ introduces # by one of the rules $i_j \to \#$; when $i_1$ and $i_2$ or $i_3$ and $i_4$ indicate the same value, then
  - for $\{Yi_1 f, \alpha x i_2 \beta f\}$, the two $i'$s become $0_1$ and $0_2$, respectively, by applying $\Phi_2$; then $\Phi_3$ makes them $\lambda$ and the process successfully simulates $i : (X \to Y, A \to x)$;
  - similarly for $\{i_3 f, \alpha x i_4 \beta f\}$, the two $i'$s become $0_3$ and $0_4$, respectively, by applying $\Phi_2$; then $f$ is deleted from both strings (using $\Phi_4$ and going to state $q_3$); $\Phi_5$ removes both $0_3$ and $0_4$ and $0_4 \to (\lambda, out)$ sends out $\alpha x \beta$;
- when either $\Phi_6$ or $\Phi_7$ is applied to $\{Xf, \alpha A\beta f\}$ this means that the use of either $(X \to Y, A_1 \to \#)$ or $(X \to Y, A_2 \to x)$ is simulated (using $\Phi_6$ for the former or $\Phi_7$ for the latter);
- once a symbol # is introduced the process never ends as $\Phi_8$ may be always triggered.

The EP system $E\Pi$ computes exactly what $G$ generates.

(ii) The EP systems $E\Pi = (\mu, X)$ with one state and two membranes is built as follows: $\mu$ is a membrane structure $[_1[_2]_2]_1$ and $X$ the Eilenberg machine:

$$X = (N_1 \cup N_2 \cup \{S, \#, f_1, f_2\} \cup \{i_j \mid 0_j \leq i_j \leq (k_3)_j, 1 \leq j \leq 4\} \cup T, T, \{q_1\}, M_1, M_2, \Phi, F, \{q_1\}),$$

where $f_1, f_2$ and $i_j$ are new symbols and:

- $M_1$ contains $Zf_1$ and $Bf_2$, with $Z$ and $B$ the symbols occurring in the right hand side of the rule of type 1, and $M_2$ is empty;
- $\Phi = \{\Phi_1, \ldots, \Phi_7\}$, is very similar with $\Phi$ built for the above proof (i), but makes use of the two membranes; $\Phi$ contains:

- $\Phi_1 = (\{X \to (Yi_1, in) \mid 1_1 \leq i_1 \leq (k_1)_1,$ and there is $i_1 : (X \to Y, A \to x)$ a matrix of type 2 $\} \cup$

$\{X \to (\lambda i_3, in) \mid 1_3 \leq i_3 \leq (k_3)_3,$ and there is $i_3 : (X \to \lambda, A \to x)$
a matrix of type 4 $\}\cup$
$\{A \to (xi_2, in) \mid 1_2 \leq i_2 \leq (k_1)_2,$ and there is $i_2 : (X \to Y, A \to x)$
a matrix of type 2 $\}\cup$
$\{A \to (xi_4, in) \mid 1_4 \leq i_4 \leq (k_3)_4,$ and there is $i_4 : (X \to \lambda, A \to x)$
a matrix of type 4 $\}\cup$
$\{U \to \# \mid U \in N_1 \cup N_2, U \notin Dom_2 \cup Dom_4\}, \emptyset)$,

- $\Phi_2 = (\emptyset, \{i_j \to i_j - 1 \mid 1 \leq j \leq 4, 0_1 < i_1 \leq (k_1)_1, 0_2 < i_2 \leq (k_1)_2,$
  $0_3 < i_3 \leq (k_3)_3,\ 0_4 < i_4 < (k_3)_4\} \cup \{f_1 \to \#, f_2 \to \#\})$,
- $\Phi_3 = (\emptyset, \{0_j \to (\lambda, out)\} \cup \{i_j \to \# \mid i_j > 0, 1 \leq j \leq 4\})$,
- $\Phi_4 = (\{f_1 \to \lambda, f_2 \to (\lambda, out)\}, \emptyset)$,
- $\Phi_5 = (\{X \to Y, A_1 \to \# \mid (X \to Y, A_1 \to \#)$ matrix of type 3 $\}\cup$
  $\{U \to \# \mid U \in N_1 \setminus Dom_3\}, \emptyset)$,
- $\Phi_6 = (\{X \to Y, A_2 \to \# \mid (X \to Y, A_2 \to \#)$ matrix of type 3 $\}\cup$
  $\{U \to \# \mid U \in N_1 \setminus Dom_3\}, \emptyset)$,
- $\Phi_7 = (\{\# \to \#\}, \{\# \to \#\})$.

In a manner very similar to the proof of (i) it may be shown that $E\Pi$ computes exactly what $G$ generates. □

EPP systems exhibit a parallel behaviour not only inside of the membrane structure but also at the underlying machine level. Potentially, all transitions emerging from active states may be triggered in one step giving birth to new cells or colliding others. One problem addressed in this case is also related to the power of these mechanisms. One may adopt the previous strategy by considering a $RE$ language generated by a matrix grammar in binary normal form to find the EPP systems computing that language. Given that EP systems have been studied in this respect, another solution would be to compare them with EPP systems. More precisely, given an EP systems with $m$ membranes, $s$ states and $p$ production rules associated via $\Phi$ components, is it possible to simulate it by an EPP system? If yes, what are the values for the number of membranes, states and rules? The next theorem gives an answer to this problem.

**Lemma 3.** *If $E\Pi$ is an EP system with $m$ membranes, $s$ states and $p$ sets of rules then there exists $E\Pi\Pi$ an EPP systems with $m' \geq m$ membranes, $s' \geq s$ states and $p' \geq p$ rule transitions such that $L(E\Pi) = L(E\Pi\Pi)$.*

*Proof.* Let $E\Pi = (\mu, X)$, be an EP systems where $\mu$ is a membrane structure consisting of $m$ membranes, and $X$ an Eilenberg machine

$$X = (V, \Gamma, Q, M_1, \ldots M_m, \Phi, F, I),$$

where $Q$ has $s$ states and $\Phi$ contains $p$ components. The following EPP system is built $E\Pi\Pi = (\mu', X')$, where $\mu' = [_0\mu]_0$ and

$$X' = (V', \Gamma, Q', M_0, M_1, \ldots M_m, \Phi', F', I),$$

with

- $V' = V \cup \{x\} \cup \{k \mid 1 \leq k \leq t\}$, where $t$ is the maximum number of transitions going out from every state of $X$;
- $Q' = Q \cup \{q_{j,0} \mid q_j \in Q\} \cup \{q_{j,k,h} \mid q_j \in Q, 1 \leq k \leq t, 1 \leq h \leq 4\}$;
- $M_0 = \{x\}$;
- $\Phi' = \Phi \cup \{\Phi_x, \Phi_{1x}, \ldots, \Phi_{tx}, \Phi_\Gamma, \Phi_{V'}\}$, where
  - $\Phi_x = (\{x \to k \mid 1 \leq k \leq t\}, \emptyset, \ldots, \emptyset)$,
  - $\Phi_{kx} = (\{k \to x\}, \emptyset, \ldots, \emptyset), 1 \leq k \leq t$,
  - $\Phi_\Gamma = (\{a \to (a, out) \mid a \in \Gamma\}, \emptyset, \ldots, \emptyset)$,
  - $\Phi_{V'} = (\{X \to (X, in) \mid X \in V' \setminus \Gamma\}, \emptyset, \ldots, \emptyset)$;
- for any $q_j \in Q$ if there are $1 \leq u \leq t$, transitions emerging from $q_j$ and $F(q_j, \Phi_{j,k}) = \{q_{j_k}\}$, $1 \leq k \leq u$ (not all $\Phi_{j,k}$ are supposed to be distinct) then the following transitions are built in $E\Pi\Pi$ :
  $F'(q_j, \Phi_x) = \{q_{j,0}\}$, $F'(q_{j,0}, \Phi_{kx}) = \{q_{j,k,1}\}$, $1 \leq k \leq u$,
  $F'(q_{j,k,1}, \Phi_{j,k}) = \{q_{j,k,2}, q_{j_k}\}$,
  $F'(q_{j,k,2}, \Phi_{V'}) = \{q_{j,k,3}\}$, $F'(q_{j,k,3}, \Phi_\Gamma) = \{q_{j,k,4}\}$, $1 \leq k \leq u$.

A computation in system $E\Pi\Pi$ proceeds in the following way: at the beginning, only the initial state is active and the memory configuration in this state is $M_0, M_1, \ldots, M_m$. If the EPP system $E\Pi$ is in a state $q_j$ and the memory configuration is $M_{j,0}, M_{j,1}, \ldots, M_{j,m}$, then $E\Pi\Pi$ must be in $q_j$ as well. We will show that always $E\Pi\Pi$ has either an active state or at most two active states but in this case one of them is $q_{j,k,h}$, $2 \leq h \leq 4$, and from the last one ($q_{j,k,4}$) the membrane configuration will vanish and possibly the EPP system sends out a string. Indeed, if $q_j$ is an active state in $E\Pi\Pi$ and $M_{j,0}, M_{j,1}, \ldots, M_{j,m}$ are its associated membrane configuration, then in one step $x$ from $M_{j,0}$ is changed by $\Phi_x$ into $k$, a value between 1 and $t$; if $u$ is the number of emerging transitions from $q_j$ in $E\Pi$, then $k > u$ implies that in the next step the current membrane configuration will vanish as no more continuation is then allowed from $q_{j,0}$; otherwise, when $1 \leq k \leq u$, only one transition may be triggered from $q_{j,0}$ and this is associated with $\Phi_{kx}$ which restores $x$ back into $M_{j,0}$ (the other transitions emerging from $q_{j,0}$ cannot be triggered). $\Phi_{kx}$ leads the EPP system into $q_{j,k,1}$. From this state there are two transitions both associated with $\Phi_{j,k}$ that are triggered in the same time and consequently $M_{j,0}, M_{j,1}, \ldots, M_{j,m}$ are processed by both $\Phi_{j,k}$'s and yield the same memory configuration $M'_{j,0}, M'_{j,1}, \ldots, M'_{j,m}$ in both $q_{j,k,2}$ and $q_{j_k}$. These are then processed in parallel, being both active states. From the former the current configuration is checked for nonterminal symbols, $\Phi_{V'}$, and then in the next step only terminal strings are sent out by using $\Phi_\Gamma$; then this memory configuration vanishes as no further transition emerges from $q_{j,k,4}$. In fact this path introduced from every $q_{j,k,1}$ has the role of collecting terminal strings outside of the system. From $q_{j,k}$ the process resumes as from $q_j$, and in two steps at most one state will be active on the path from $q_j$. In this way $E\Pi$ and $E\Pi\Pi$ compute the same language, thus $L(E\Pi) = L(E\Pi\Pi)$. □

*Note 1.* From Lemma 3 it follows that if the EP system $E\Pi$ has $m$ membranes, $s$ states, $p$ components of $\Phi$, and the maximum number of transitions emerging from every state is $t$ then the equivalent EPP system has $m' = m+1$ membranes, at most $s' = (2 + 4t)s$ states, and at most $p' = p + t + 3$ sets of rules.

**Theorem 4.** $EPP_{2,54,15} = EPP_{3,30,17} = RE$.

*Proof.* By using Note 1 and the constructions from the proof of Theorem 3 the result follows. □

Obviously lower bounds may be obtained for the above discussed parameters when the two constructions from the proof of Theorem 3 are used in order to get an EPP equivalent system. This is achieved by applying the procedure provided by the proof of Lemma 3, but it is left as an exercise for the reader.

## 4 Linear Solution to SAT Problem

SAT (satisfiability of propositional formulae in conjunctive normal form) is a well known NP-complete problem. This problem asks whether or not, for a given formula in the conjunctive normal form, there is a truth-assignment of the variables such that it becomes true. So far some methods to solve in polynomial or just linear time this problem have been indicated, but at the expense of using an exponential space of values.

**Theorem 5.** *The SAT problem can be solved in a time linear in the number of variables and the number of clauses by using an EPP system.*

*Proof.* Let $\gamma$ be a formula in conjunctive normal form consisting of $m$ clauses, $C_1, \ldots, C_m$, each one being a disjunction, and the variables used are $x_1, \ldots, x_n$. The following EPP system, $E\Pi\Pi = (\mu, X)$, may be then constructed:

$$\mu = [_1[_2 \ldots [_{m+1}]_{m+1} \ldots]_2]_1,$$

$$X = (V, \Gamma, Q, M_1, \ldots M_{m+1}, \Phi, F, I),$$

where:

- $V = \{a_k, t_k, f_k \mid 1 \leq k \leq n\}$;
- $\Gamma = \{t_k, f_k \mid 1 \leq k \leq n\}$;
- $Q = \{q_1, q_2\}$;
- $M_1 = \ldots = M_m = \emptyset$, $M_{m+1} = \{a_1\}$;
- $\Phi = \{\Phi_1, \ldots, \Phi_5\}$;
  - $\Phi_1 = (\emptyset, \ldots, \emptyset, \{a_k \to f_k a_{k+1} \mid 1 \leq k \leq n-1\})$,
  - $\Phi_2 = (\emptyset, \ldots, \emptyset, \{a_k \to t_k a_{k+1} \mid 1 \leq k \leq n-1\})$,
  - $\Phi_3 = (\emptyset, \ldots, \emptyset, \{a_n \to (f_n, out)\})$,
  - $\Phi_4 = (\emptyset, \ldots, \emptyset, \{a_n \to (t_n, out)\})$,
  - $\Phi_5 = (\ \{t_k \to (t_k, out) \mid x_k$ is present in $C_1, 1 \leq k \leq n\} \cup$
    $\{f_k \to (f_k, out) \mid \neg x_k$ is present in $C_1, 1 \leq k \leq n\}$,
    $\ldots$,
    $\{t_k \to (t_k, out) \mid x_k$ is present in $C_m, 1 \leq k \leq n\} \cup$
    $\{f_k \to (f_k, out) \mid \neg x_k$ is present in $C_m, 1 \leq k \leq n\}$, $\emptyset)$;
- $F(q_1, \Phi_k) = \{q_1\}$, $1 \leq k \leq 2, F(q_1, \Phi_k) = \{q_2\}$, $3 \leq k \leq 4, F(q_2, \Phi_5) = \{q_2\}$;
- $I = \{q_1\}$.

$E\Pi\Pi$ starts from state $q_1$ with $\emptyset, \ldots, \emptyset, \{a_1\}$. By applying $n-1$ times $\Phi_1$ and $\Phi_2$ in parallel and then $\Phi_3$ and $\Phi_4$ one generates all truth values for the n variables in the form of $2^n$ strings with $t_k$ or $f_k$ indicating that variable $x_k$ is either *true* or *false*. All these combinations are obtained in $n$ steps in state $q_2$. In the next $m$ steps $\Phi_5$ checks whether or not at least one truth-assignment satisfies all clause; this, if exists, will exit the system. The SAT problem is solved in this way in $n + m$ steps. □

## 5 Conclusions

In this paper two types of Eilenberg P systems, namely EP systems and EPP systems, have been introduced. They combine the control structure of an Eilenberg machine as a driven mechanism of the computation with a cell-like structure having a hierarchical organisation of the objects involved in the computational process. The computational power of EP systems is investigated in respect of three parameters: number of membranes, number of states, and number of sets of rules. It is proved that when only one state and one set of rules are used, four membranes suffice to compute $MAT$ languages. It may be easily observed that in this case the number of states is irrelevant as with only one single set of rules, even distributed across many states, one cannot compute more than with a single state and the same set of rules, i.e., $EP_{m,s,1} = MAT$, $m \geq 4, s \geq 1$. When at least three states and eight sets of rules or two membranes and seven sets of rules are used, then the whole set of $RE$ languages may be computed, i.e., $EP_{1,3,8} = RE$ or $EP_{2,1,7} = RE$. A number of questions regarding lower limits for the above parameters remain to be further addressed. It is possible to compute $RE$ by using EP systems with less than three states and/or eight sets of rules, i.e., $EP_{1,s,p}$, where $s < 3$ and/or $p < 8$? Is it true that $EP_{1,1,7} = RE$, $p < 7$?

EPP systems represent the parallel counter-part of EP systems, allowing not only the rules inside of the cell-like structure to develop in parallel, but also the transitions emerging from the same state. More than this, all states that are reached during the computation process as target states, may trigger in the next step all transitions emerging from them. It is shown that a general method to simulate an EP system as an EPP system computing the same language may be stated. This result allows us to map all properties concerning computationally completeness properties of EP systems onto EPP systems. Apart from the fact that EPP systems might describe interesting biological phenomena like cell division and collision, it is also a computationally complete device and an effective mechanism for solving NP-complete problems, like SAT, in linear time.

## References

1. T. Bălănescu, T. Cowling, H. Georgescu, M. Gheorghe, M. Holcombe, C. Vertan, Communicating stream X-machines systems are no more than X-machines, *J. Universal Comp. Sci.,* 5, 9 (1999), 494–507.

2. C. Calude, Gh. Păun, *Computing with Cells and Atoms,* Taylor and Francis, London, 2000.
3. E. Csuhaj-Varju, J. Dassow, J. Kelemen, Gh. Păun, *Grammar Systems. A Grammatical Approach to Distribution and Cooperation,* Gordon & Breach, London, 1994.
4. J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory,* Springer Verlag, Berlin, 1989.
5. S. Eilenberg, *Automata, Languages and Machines,* Academic Press, 1974.
6. C. Ferretti, G. Mauri, Gh. Păun, C. Zandron, On Three Variants of Rewriting P Systems, *Pre-proceedings of Workshop on Membrane Computing (WMC-CdeA2001)*, (C. Martin-Vide, Gh. Păun, eds), Curtea de Argeş, Romania, August 2001, 63–76, and *Theor. Comp. Sci.,* to appear.
7. R. Freund, Gh. Păun, On the number of non-terminals in graph-controlled, programmed, and matrix grammars, *Proc. Conf. Universal Machines and Computations,* Chisinau, 2001 (M. Margenstern and Y. Rogozhin, eds.), Springer-Verlag, Berlin, 2001.
8. M. Gheorghe, Generalised stream X-machines and cooperating grammar systems, *Formal Aspects of Computing,* 12 (2000), 459–472.
9. M. Gheorghe, M. Holcombe, P. Kefalas, Computational models of collective foraging, *BioSystems,* 61 (2001), 133–141.
10. M. Holcombe, X-machines as a basis for dynamic system specification, *Software Engineering Journal,* 3, 2 (1988), 69–76.
11. M. Holcombe, Computational models of cells and tissues: Machines, agents and fungal infection, *Briefings in Bioinformatics,* 2 (2001), 271–278.
12. M. Holcombe, What are X-machines, *Formal Aspects of Computing,* 12 (2000), 418–422.
13. M. Holcombe, F. Ipate, *Correct Systems Building a Business Process Solution,* Springer, Applied Computing Series, 1998.
14. K. Jensen, *Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use,* vol 1-3, Springer, Berlin, 1992, 1994, 1997.
15. S. Ji, The Bhopalator, An information/energy dual model of the living cell, *Pre-proceedings of Workshop on Membrane Computing (WMC-CdeA2001)* (C. Martin-Vide, Gh. Păun, eds), Curtea de Argeş, Romania, August 2001, 123–141, and *Fundamenta Informaticae,* 49 (2002), 147–165.
16. P. Kefalas, Formal modelling of reactive agents as an aggregation of simple behaviours, LNAI 2308 (I.P. Vlahavas, C.D. Syropoulos, eds.), Springer, 461–472, 2002.
17. Z. Kohavi, *Switching and Finite Automata Theory,* McGraw-Hill, 1978.
18. S.N. Krishna, R. Rama, On the power of P systems with sequential and parallel rewriting, *Intern. J. Computer Math.,* 77, 1-2 (2000), 1–14.
19. S.N. Krishna, R. Rama, P systems with replicated rewriting, *Journal of Automata, Languages and Combinatorics,* 6 (2001), 345–350.
20. A. Păun, P systems with string objects: Universality results, *Pre-proceedings of Workshop on Membrane Computing (WMC-CdeA2001)* (C. Martin-Vide, Gh. Păun, eds), Curtea de Arges, Romania, August 2001, 229–241.
21. Gh. Păun, Computing with membranes, *Journal of Computer System Sciences,* 61, 1 (2000), 108–143 (see also *Turku Center for Computer Science, TUCS Report No* 208, 1998, http://www.tucs.fi).
22. Gh. Păun, *Membrane Computing. An Introduction,* Springer, Berlin, 2002.

# A MzScheme Implementation of Transition P Systems

Delia Balbontín Noval, Mario J. Pérez Jiménez,
and Fernando Sancho Caparrini

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla, España
{Delia.Balbontin,Mario.Perez,Fernando.Sancho}@cs.us.es

**Abstract.** The main goal of this paper is to present the design of an MzScheme program that allows us to simulate the behavior of transition P systems. For that, a library of procedures have been developed that work in two stages. In the first one, the *parsing/compiling* stage, the input P system is checked, and if it is well defined, then it is represented by means of an internal grammar. In a second stage, the *simulation*, the computation tree associated to the P system is generated until a prefixed level.

## 1 Introduction

In October 1998, Gheorghe Păun [1] introduced a new computability model of a non-deterministic and highly parallel type, the *membrane systems*. They are based on the synchronized work of several units, called *membranes*, structured in a dynamic hierarchy (understood as vesicles in a space) embedded in a *skin membrane* that separates the system from the environment. When a membrane has no membrane inside, it is called *elementary*. Each membrane encloses a space between it and the membranes directly included in it (if any). This space (the *region* of the membrane) can contain a multiset (a set where the elements can be repeated) of objects (represented by symbols of a given alphabet) and a set of (evolution) rules for them. Each membrane defines an unique region.

This model, called *transition P systems*, is inspired from the observation that the processes which take place in the complex structure of a living cell can be viewed as computation–like processes.

We present here a library of MzScheme procedures [5], that allows us both to input easily a transition P system and to simulate its non-deterministic and highly parallel behavior. It reads, analyzes and compiles the input data defining a P system; then, it generates the subsequent computations.

Our implementation is based on the formalization given in [3].

The program runs in two independent stages: *parsing/compiling* and *simulation/running*.

$$\Pi - input \quad \xrightarrow{parser/compiler} \quad \Pi \quad \xrightarrow{simulator} \quad \mathbf{Comp}(\Pi)$$

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 58–73, 2003.

At stage one, *parsing/compiling*, the input data are read and the respective P system is rewritten as an element of the language generated by a proposed *internal grammar*. To get it, the input data have to be *syntactically correct* according to the *input grammar*. Moreover, they have to define a *well defined* P system (according to the formalization above mentioned).

Stage two, *simulation/running*, starts when the *parsing/compiling* is finished. Starting from the P system *initial configuration*, the associated *computation tree* is generated. The expansion of that *computation tree* is made in a progressive way, level by level (*breadth expansion*), until to a given depth level. To get it we follow a *breadth-expansion-tree* scheme based on the definitions and steps proposed in [3]:

*applicable-rules* $\rightarrow$
*applicability-vectors* $\rightarrow$
*applicability-matrices* $\rightarrow$ *configurations*

This paper is organized as follows: Section 2 briefly presents some basic concepts about a formalization of transition P systems, following [3]. Section 3 describes briefly the whole *simulator* scheme. Section 4 is about the way to input a P system, showing the proposed *input grammar*. Section 5 presents the *internal grammar* and describes the *parser/compiler* performance. Section 6 describes the *simulator* behavior properly. Finally, in Section 7 we present a complete example to illustrate the way of working of the program.

## 2 Preliminaries about a Formalization of Transition P Systems

Following [3], we recall here the basic concepts and definitions about P systems.

### 2.1 Membrane Structure and Cells

A *membrane structure* is a rooted tree, where the nodes are called *membranes*, the root is called *skin*, and the leaves are called *elementary membranes*.

A *cell* over an alphabet, $A$, is a pair $(\mu, M)$, where $\mu = (V(\mu), E(\mu))$ is a membrane structure, and $M$ is an application, $M : V(\mu) \longrightarrow \mathbf{M}(A)$ (the set of multisets over $A$).

### 2.2 Evolution Rules

Let $C = (\mu, M)$ be a cell over an alphabet $A$. Let $x \in V(\mu)$. An *evolution rule* associated to $x$ is a 3-tuple $r = (d_r, v_r, \delta_r)$ where $d_r$ is the left-side of the rule, $v_r$ is the right-side of the rule, and $\delta_r \in \{\neg\delta, \delta\}$ indicates if the application of the rule dissolves the membrane.

A *collection $R$ of evolution rules* associated to $C$ is a function with domain $V(\mu)$ such that for every membrane $x \in V(\mu)$, $R_x = \{r_{x,1}, \ldots, r_{x,s_x}\}$ is a finite

set (possibly empty) of (evolution) rules associated to $x$. A *priority relation over* $R$ is a function, $\rho$, with domain $V(\mu)$ such that for every membrane $x \in V(\mu)$, $\rho_x$ is a strict partial order over $R_x$ (possibly empty).

### 2.3 Transition P Systems

A *transition P system* is a 4-tuple $\Pi = (A, C_0, \mathcal{R}, i_0)$, where:

- $A$ is a non-empty finite set (usually called base alphabet).
- $C_0 = (\mu_0, M_0)$ is a cell over $A$.
- $\mathcal{R}$ is an ordered pair $(R, \rho)$ where $R$ is a collection of (evolution) rules associated to $C_0$, and $\rho$ is a priority relation over $R$.
- $i_0$ is a node of $\mu_0$, which specifies the output membrane of $\Pi$.

The number $|V(\mu_0)|$ is called the *degree* of $\Pi$.

### 2.4 Configurations

A *configuration*, $C$, of a P system, $\Pi = (A, C_0, \mathcal{R}, i_0)$ with $C_0 = (\mu_0, M_0)$, is a cell $C = (\mu, M)$ over $A$, where $V(\mu) \subseteq V(\mu_0)$, and $\mu$ has the same root as $\mu_0$. The configuration $C_0$ will be called the *initial configuration of* $\Pi$.

### 2.5 Applicability

Let $x \in V(\mu_0)$. We say that the (evolution) rule $r \in R_x$ is *semi-applicable* to $C$ if the membrane associated to node $x$ exists in $C$ (dissolution is not allowed in the root node), the membrane associated to $x$ has all the necessary objects to apply the rule, and nodes where the rule tries to send objects (by means of $in_y$) are children of $x$.

We say that the rule $r \in R_x$ is *applicable* to $C$, if it is semi-applicable to $C$ and there is no semi-applicable rules in $R_x$ with higher priority.

We say that $\boldsymbol{p} \in \mathbf{N}^{\mathbf{N}}$ is an *applicability vector* over $x \in V(\mu)$ for $C$, and we will denote it as $\boldsymbol{p} \in \mathbf{Ap}(x, C)$, if it has correct size (that is, for all $j$ greater the number of rules associated to $x$ we have $p(j) = \emptyset$), every rule can be applied as many times as the vector $\boldsymbol{p}$ indicates, all the rules can be applied simultaneously, and it is maximal.

We will say that $P : V(\mu_0) \longrightarrow \mathbf{N}^{\mathbf{N}}$ is an *applicability matrix* over $C$, denoted $P \in \mathbf{M}_{\mathbf{Ap}}(C)$, if for every $x \in V(\mu_0)$ we have that $P(x) \in \mathbf{Ap}(x, C)$.

### 2.6 Transitions

The *execution* of $P$ over $C = (\mu, M)$, denoted $P(C)$, returns a new configuration $C' = (\mu', M')$ of $\Pi$, that can be considered acting in two stages: $(\mu, M) \rightarrow (\mu, M'') \rightarrow (\mu', M')$.

In the first stage we suppose that the rules are applied without attending dissolving actions, and in the second one dissolution and distribution of contents are carried out.

We will say that a configuration $C_1$ of a $P$ system $\Pi$ yields a configuration $C_2$ by a *transition in one step* of $\Pi$, denoted $C_1 \Rightarrow_\Pi C_2$, if there exists a non–zero applicability matrix over $C_1$, $P$, such that $P(C_1) = C_2$.

### 2.7 Computation Tree

The *computation tree of a P system* $\Pi$, denoted $\mathbf{Comp}(\Pi)$, is a rooted labeled maximal tree defined as follows: the root of the tree is the initial configuration, $C_0$, of $\Pi$; the children of a node are the configurations that follow in one step of transition; nodes and edges are labeled by configurations and applicability matrices, respectively, in such way that two labeled nodes $C, C'$ are adjacent in $\mathbf{Comp}(\Pi)$, by means of an edge labeled with $P$, if and only if $P \in \mathbf{M_{Ap}}(C) - \{\mathbf{0}\}$ and $C' = P(C)$. The maximal branches of $\mathbf{Comp}(\Pi)$ will be called *computations* of $\Pi$. We will say that a computation of $\Pi$ *halts* if it is a finite branch. The configurations verifying $\mathbf{M_{Ap}}(C) = \{\mathbf{0}\}$ will be called *halting configurations*.

## 3 Preliminaries about the P Systems Simulator

We consider that the basic features of a computing program able to simulate transition P systems should be the following:

1. To have a formal definition of transition P systems to be based on.
2. To choose a suitable programming language to implement the simulation.
3. To have an easy way to input the data describing the P system.
4. To choose an efficient internal representation of P systems.
5. To design a parser/compiler to analyze the input data and to obtain the P system internal representation.
6. To design a P system simulator of computations to generate the respective computation tree.

As we said previously, the implementation we present here has been developed on MzScheme (a functional language from Lisp family), and it is based on the formalization given in the above section, but slightly modified. This modification arises from the convenience to identify the *applicable* rules to a given configuration.

The rules of a P system are static elements. Nevertheless, to determine if a rule $r = (d_r, v_r, \delta_r)$ is *applicable* to an arbitrary configuration $C$, a new component $\alpha_r \in \{\#\mathtt{t}, \#\mathtt{f}\}$ has been added, getting $r^* = (d_r, v_r, \delta_r, \alpha_r)$. Initially, $\alpha_r$ will be set to $\#\mathtt{f}$; it will be modified to $\#\mathtt{t}$ if (and only if) the rule $r$ is applicable to $C$. Consequently, if we denoted for every $x \in V(\mu_0)$, $R_x = \{r_{x,1}, \ldots, r_{x,s_x}\}$, then we have $R_x^* = \{r_{x,1}^*, \ldots, r_{x,s_x}^*\}$, with $r_{x,j}^* = (d_{x,j}, v_{x,j}, \delta_{x,j}, \alpha_{x,j})$ and $\alpha_{x,j} = \#\mathtt{f}$; then, $R^* = \bigcup_{x \in V(\mu_0)} R_x^*$, and $\mathcal{R} = (R^*, \rho)$.

Moreover, for every configuration, $C = (\mu, M)$ and every $x \in V(\mu_0)$, we will denote by $R_x^C = \{r_{x,1}^C, \ldots, r_{x,s_x}^C\}$, with $r_{x,j}^C = (d_{x,j}, v_{x,j}, \delta_{x,j}, \alpha_{x,j})$ and $\alpha_{x,j} = \#\mathtt{t}$ if and only if the rule $r_{x,j}$ is applicable to $C$, the *tagged-rules* of $x$ to $C$. Finally, we will note $R^C = \bigcup_{x \in V(\mu_0)} R_x^C$.

# 4 The Input of a Transition P System

To define a P system we need to input its membrane structure and describe the content of every membrane. Each membrane has symbols from a given alphabet, transition rules and priority relations over them. The membrane structure has to be a rooted tree and the priority between rules must be a strict partial order.

## 4.1 Default Settings

In order to introduce easily any P system we have considered, by default, that:

1. Only finite alphabets $A$ will be used, and the elements of $A$ are *symbols*.
2. A *word* $\in A^*$ is a string of *symbols* of $A$. We will represent the empty word by ().
3. The membranes will be labeled with the *the first* $N$ *natural numbers*, where $N$ is the degree of the P system.
4. The skin membrane is labeled with 1.
5. A distinguished membrane is considered as the *output membrane*.
6. We will input the membrane structure of a P system as a list of *contain-pairs* $(i\ j)$, representing the relation "*membrane* $i$ *contains membrane* $j$".
7. Every rule has a *word* as its antecedent, and a set of *actions* as its consequent. Only the last action could be "**delete**". The other ones have the form (*word target*).
8. A *target* could be "**here**", "**out**" or a membrane label.
9. If a membrane has $k > 0$ rules, then their labels go from 1 to $k$.
10. We represent the relation "*rule* $r$ *runs before rule* $s$" by the *preference-pair* $(r\ s)$.
11. Every membrane contains a *word*, a list of rules, and a list of *preference-pairs*.

## 4.2 The Input Grammar

With the default settings provided above, any P system of degree $N$, over an alphabet $A$, is recognized by the *input grammar* defined as follows:

$$
\begin{array}{ll}
< input - ps > & ::= (A\ N < struct >< objects >< rules >< orders >< output >) \\
< struct > & ::= (< arc >< arc > \cdots < arc >) \\
< arc > & ::= (< memb - ref >< memb - ref >) \\
< memb - ref > & ::= 1 \mid 2 \mid 3 \mid \ldots \mid N \\
 & \\
< objects > & ::= [< word >< word >.\overset{N}{\ldots}.< word >] \\
< word > & ::= \forall\ w \in A^* \\
 & \\
< rules > & ::= [< memb - rules >.\overset{N}{\ldots}.< memb - rules >] \\
< memb - rules > & ::= (< rule >< rule > \cdots < rule >) \\
< rule > & ::= (< word > \rightarrow\ (< action > \cdots < action > \textbf{delete})) \mid \\
 & \quad\ (< word > \rightarrow\ (< action > \cdots < action >))
\end{array}
$$

$$
\begin{array}{ll}
< action > & ::= (< word >< target >) \\
< target > & ::= \textbf{here} \ \mid \ \textbf{out} \ \mid \ < memb - ref >
\end{array}
$$

$$
\begin{array}{ll}
< orders > & ::= [< memb - or >< memb - or >.\overset{N}{\ldots}.< memb - or >] \\
< memb - or > & ::= (< pref - pair >< pref - pair > \cdots < pref - pair >) \\
< pref - pair > & ::= (< rule - ref >< rule - ref >) \\
< rule - ref > & ::= 1 \mid 2 \mid 3 \ldots \\
< output > & ::= < memb - ref >
\end{array}
$$

Here, $(a\ b \ldots z)$ stands for a *list* (standard MzScheme *list*), and $[a\ b\ \overset{N)}{\ldots}\ z]$ stands for a *vector* of $N$ elements (standard MzScheme *vector*).

## 5 The Parser/Compiler

The parser/compiler reads the input data describing a P system and analyzes: if they are *syntactically* correct according to the *input grammar*, if they define a *well defined* P system according to the chosen formalization, and, if no error appears, it returns the P system according to the proposed *internal grammar*.

Even if the input system is syntactically correct, we cannot conclude that any input data recognized by the *input grammar*, define a *well-defined* P system. In fact, it could happen that the structure $< struct >$ defined as a list of arcs $(< arc >^*)$ were not a *rooted tree* with root at membrane label 1; or, that there exists a membrane, such that the order relation $(< mem - or >)$ defined as a list of preference pairs $(< pref - pair >^*)$ were not a strict partial order.

The MzScheme sentence to execute the parser/compiler is:

`(parser-ps` $N$ $A$ `<struct> <objects> <rules> <orders> <output>)`

This process of parsing/compiling works as follows:

- The alphabet $A$ is checked.
- The *rooted tree* $\mu$, associated to the membrane structure, is created.
- For every membrane $x$, its objects are encoded as a *multiset* $M_x$, getting $M : V(\mu) \longrightarrow \mathbf{M}(A)$.
- Then, the *initial configuration*, $C_0 = (\mu, M)$, is built.
- Every rule, $r$, from the input data is encoded by $r^* = (d_r,\ v_r,\ \delta_r,\ \alpha_r)$, where $\alpha_r$ is set initially to `#f`. Then, one gets $R^*$.
- For every membrane $x$ a strict partial order $\rho_x : R_x \times R_x \longrightarrow \{\#\mathtt{t}, \#\mathtt{f}\}$ is returned, with: $\rho_x(r,t) = \#\mathtt{t} \Leftrightarrow$ " $r$ runs before $s$" at $x$. So, we obtain $\rho$.
- From $R^*$ and $\rho$ we have $\mathcal{R} = (R^*, \rho)$.
- The *output* membrane is checked to be in $V(\mu)$, getting $i_0$.

If no error occurs, the `parser-ps` procedure returns a *well-defined* P system $\Pi = (A, C_0, \mathcal{R}, i_0)$ as an element recognized by the *internal grammar* below.

### 5.1 Internal Grammar

The grammar to represent internally and to deal with P systems of degree $N$ is the following:

$$
\begin{array}{ll}
< ps > & ::= [< alph > ; < conf > ; < Rules > ; < orders > ; < output >] \\
 & \\
< alph > & ::= [\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_K}] \\
 & \\
< conf > & ::= [< tree > ; < multisets >] \\
 & \\
< tree > & ::= [< vertices > ; < arcs > ; < root >] \\
< vertices > & ::= \{< x > , \ldots , < x >\} \\
< arcs > & ::= \{< arc > , < arc > , \ldots , < arc >\} \\
< arc > & ::= [< x > ; < x >] \\
< x > & ::= \forall\, n \in \mathbf{N}^+ \mid n \leq N \\
< root > & ::= 1 \\
 & \\
< multisets > & ::= [< multiset >< multiset > \overset{N)}{\ldots} < multiset >] \\
< multiset > & ::= [< nat >< nat > \overset{\mathbf{K})}{\ldots} < nat >] \\
< nat > & ::= \forall\, n \in \mathbf{N} \\
 & \\
< Rules > & ::= [< rules >< rules > \overset{N)}{\ldots} < rules >] \\
< rules > & ::= [< rule >< rule > \ldots < rule >] \\
< rule > & ::= [< anteced > ; < actions > ; < dissol > ; < app - tag >] \\
< anteced > & ::= < multiset > \\
< actions > & ::= (< action >< action > \cdots < action >) \\
< action > & ::= [< multiset > ; < target >] \\
< target > & ::= \mathbf{here} \mid \mathbf{out} \mid < x > \\
< dissol > & ::= \#\mathtt{t} \mid \#\mathtt{f} \\
< app - tag > & ::= \#\mathtt{t} \mid \#\mathtt{f} \\
 & \\
< orders > & ::= [< test >< test > \overset{N)}{\ldots} < test >] \\
< test > & ::= \lambda : rules \times rules \longrightarrow \{\#\mathtt{t}, \#\mathtt{f}\} \\
 & \\
< output > & ::= < x >
\end{array}
$$

## 6 The Simulator

Once the *parsing/compiling* task is finished, we have a *well-defined* P system, namely $\Pi = (A, C_0, \mathcal{R}, i_0)$, and we have to generate the computation tree $\mathbf{Comp}(\Pi)$. To do that we use the procedure `configurations`:

$$\Pi \quad \xrightarrow{\texttt{configurations}} \quad \mathbf{Comp}(\Pi)$$

We get the computation tree $\mathbf{Comp}(\Pi)$ through the MzScheme sentence `(configurations` $\Pi$ *level*`)`.

The procedure `configurations` is based on the `breadth-expansion-tree` procedure that, starting from the initial configuration $C_0$, generates level by level the *computation tree*. It uses the auxiliary procedures `applicability-vectors`, `tag-rules` and `apply-matrix`. Here we present a brief outline. We will give in the next sections a detailed description of every one.

The operators to compute the *successor configurations* of a given configuration, $C$, are the *applicability matrices*. The process to generate the elements of $\mathbf{M_{A_P}}(C)$ works as follows:

- $R^C$ (that is, the *tagged-rules* of $x$ to $C$) is obtained by the `tag-rules` procedure. For every rule $r^* = (d_r, v_r, \delta_r, \alpha_r) \in R^*$, it sets $\alpha_r = \#\mathtt{t}$ iff $r$ is applicable to $C$.
- Every $R_x^C$, for every membrane $x$ in $C$, is easily obtained from $R^C$.
- Every $\mathbf{Ap}(x, C)$ (that is, the *applicability vectors* of membrane $x$ in $C$) is generated from $R_x^C$, by means of the `applicability-vectors` procedure.
- Finally, $\mathbf{M_{Ap}}(C)$ is constructed as a cartesian product from the set of *applicability vectors* $\mathbf{Ap}(x, C)$, of every membrane $x$ in $C$.

$$C \xrightarrow{\texttt{tag-rules}} R^C \begin{array}{l} \overset{x_1}{\nearrow} \\ \xrightarrow{\texttt{x}_2} \\ \overset{x_3}{\searrow} \end{array} \begin{array}{l} R_{x_1}^C \xrightarrow{\texttt{applicability-vectors}} \mathbf{Ap}(x_1, C) \searrow \\ R_{x_2}^C \xrightarrow{\texttt{applicability-vectors}} \mathbf{Ap}(x_2, C) \rightarrow \mathbf{M_{A_P}}(C) \\ R_{x_3}^C \xrightarrow{\texttt{applicability-vectors}} \mathbf{Ap}(x_3, C) \nearrow \end{array}$$

Then, every $P \in \mathbf{M_{Ap}}(C)$ is applied to $C$ to obtain the *successor configuration* $P(C)$. To do that the `apply-matrix` procedure is used.

$$\mathbf{M_{A_P}}(C) \rightarrow \begin{array}{l} \nearrow P_1 \in \mathbf{M_{A_P}}(C) \xrightarrow{\texttt{apply-matrix}} C_1 = P_1(C) \\ P_2 \in \mathbf{M_{A_P}}(C) \xrightarrow{\texttt{apply-matrix}} C_2 = P_2(C) \\ \searrow P_3 \in \mathbf{M_{A_P}}(C) \xrightarrow{\texttt{apply-matrix}} C_3 = P_3(C) \end{array}$$

### 6.1 The Breadth-Expansion-Tree Procedure

This procedure is based on a *dynamic* breadth-search scheme; this means that for every node of the tree to be built, the applicable operators are generated *dynamically*.

To start, the `breadth-expansion-tree` procedure needs: (1) an *initial node*, $n_0$, (2) a test `final-node?` to check if a node $n$ is or not a *final node*, (3) a function `generate-op`, that, taking a node $n$, returns the set of operators $Op_n$ to be applied to $n$ and, finally, (4) another function `apply-op` that, taking a node $n$ and an operator $op \in Op_n$, returns the *successor node* of $n$ by this operator $op$.

The `breadth-expansion-tree` procedure expands the tree and returns the set of *final nodes*.

```
Procedure breadth-expansion-tree (n_0 final-node? generate-op
apply-op)
final-nodes ← {}
open-nodes ← {n_0}
Repeat until open-nodes = ∅ do
n ← the first node in open-nodes
succ_n ← {}
If (final-node? n) = #t
  then final-nodes ← {n}∪ final-nodes
  else
       Op_n ← (generate-op n)
       For every op ∈ Op_n do
           suc ← (apply-op op n)
           If suc ≠ #f ∧ suc ∉ open-nodes then
             succ_n ← succ_n ∪ {suc}
open-nodes ← (open-nodes−{n}) ∪ succ_n
Return final-nodes
```

The procedures `configurations` and `applicability-vectors`, to generate configurations and applicability vectors, respectively, are based on this procedure.

### 6.2 The Configurations Procedure

For a given P system $\Pi = (A, C_0, \mathcal{R}, i_0)$, we generate $\mathbf{Comp}(\Pi)$ (until a level given by the user), through the MzScheme sentence `(configurations` $\Pi$ *level*`)`. This procedure works as follows:

1. It starts defining locally:
   - The *node-structure* as $<$ `node` $>$ ::= $[C; R^C; path_C]$, where $C$ ia a configuration; $R^C$, the *tagged-rules* for $C$; and $path_C$, the list of operators applied to reach the actual node from the initial one.
   - The `final-node?` test. A node $n = [C; R^C, path]$ is a *final node* if either it is a *halting node*, or the *path* length has reached the value of *level*.
   - The `generate-op` function. It takes a node $n = [C; \mathbf{R}^C; path]$ and returns the *applicability matrices* $\mathbf{M_{Ap}}(C)$. It needs the procedure `applicability-vectors`.
   - Finally, the procedure `apply-op`, which, taking a node $n = [C; R^C, path]$ and an applicability matrix $P \in \mathbf{M_{Ap}}(C)$, returns the *successor node* $n' = [C'; R^{C'}; P \cup path]$. It needs the procedures `apply-matrix` and `tag-rules`.
2. Then, it builds the *init-node*: $n_0 = [C_0; R^{C_0}; ()]$, making use of the procedure `tag-rules` to get $R^{C_0}$.

3. It expands the tree through the sentence:
   `(breadth-expansion-tree`
   *init − node* `final-node? generate-op apply-op)`
4. Finally, it returns the list of *final-nodes* $[C; R^C; path_C]$.

`Procedure configurations` (*Π level*)

1. *Local definitions*
   $< \texttt{node} >$ $::= [C; R^C; path_C]$
   $\texttt{final} - \texttt{node?}$ $::= \lambda_1 : < \texttt{node} > \longrightarrow \{\#\texttt{t}, \#\texttt{f}\}$
   $\texttt{generate} - \texttt{op}$ $::= \lambda_2 : < \texttt{node} > \longrightarrow \mathbf{M}_{\mathbf{A_p}}(C) = (P_1, P_2, \dots)$
   $\texttt{apply} - \texttt{op}$ $::= \lambda_3 : k \times < \texttt{node} > \longrightarrow [C'; R^{C'}; path_{C'}]$
   with $C' = P_k(C)$
   and, $path_{C'} = P_k \cup path_C$
2. *The initial node*
   $R^{C_0} \leftarrow (\texttt{tag} - \texttt{rules}\ \texttt{C}_0\ \texttt{R}^*\ \rho)$
   $path_{C_0} \leftarrow ()$
   $n_0 \leftarrow [C_0; R^{C_0}; path_{C_0}]$
3. *The final-nodes*
   *final-nodes* $\leftarrow$ `(breadth-expansion-tree`
   $n_0$ `final-node? generate-op apply-op)`
4. `Return` *final-nodes*

*Notes:*

- $\lambda_1([C; R^C; path_C]) = \#\texttt{t} \leftrightarrow (\alpha_r = \#\texttt{f}\ \forall r \in R^C) \vee |path_C| = level$
- $\lambda_2$ uses `applicability-vectors` procedure to get $\mathbf{M}_{\mathbf{A_p}}(C)$.
- $\lambda_3$ uses `apply-matrix` procedure to get $C' = P(C)$ and then, `tag-rules` to get $R^{C'}$.
- Every node $[C; R^C; path_C] \in$ *final-nodes*, contains all the information we need about the *computation tree*. Particularly,
  - If for every $r \in R^C$ is $\alpha_r = \#\texttt{f}$, then $C$ is a *halting configuration*, and $path_C$ is a *halting computation* of $\Pi$.
  - Otherwise, $C$ is a *non-halting configuration*, and the branch $path_C$ could be extended further than the prefixed *level*.

### 6.3 The Applicability-Vectors Procedure

To generate the *applicability vectors* for a membrane $x$ in $C$, we only need $M_x$ and $D = [d_1, d_2, \dots, d_{s_x}]$, where $M_x$ is the *multiset* of $x$, and $d_r$ $(r = 1, 2, \dots, s_x)$ is the antecedent of the *tagged*-rule $r$ in $R_x^C$, provided that $\alpha_r = \#\texttt{t}$. (*Note*: if $\alpha_r = \#\texttt{f}$, then we take $d_r = \#\texttt{f}$.) We generate $\mathbf{Ap}(x, C)$ through the MzScheme sentence: `(applicability-vectors` $M_x$ $D$`)`. The procedure works as follows:

1. It starts defining locally:
   - The *node-structure* as $<$ `node` $>$ $::= [m; V]$, where $m$ is a multiset, and $V = [v_1, v_2, \ldots, v_{s_x}]$.
   - The `final-node?` test. A node $n = [m; V]$ is a final node if $\forall\, d_r \in D$ $(d_r = \#\mathtt{f} \vee m < d_r)$.
   - The `generate-op` function. It returns the operators list $(d_1, d_2, \ldots, d_{s_x})$.
   - The `apply-op` procedure. From a node $n = [m; V]$, and an operator $d_r \neq \#\mathtt{f}$, it returns the *successor node* $n' = [m'; V']$, with $m' = m - d_r$, $v'_r = v_r + 1$ , and $v'_j = v_j$, $\forall\ j \neq r$. If $d_r = \#\mathtt{f}$, then it returns $\#\mathtt{f}$.
2. Then, it builds the *init-node*: $n_0 = [M_x; [0, 0, \overset{s_x}{\ldots}, 0]]$.
3. It expands the tree through the sentence:
   `(breadth-expansion-tree`
   $init - node$ `final-node? generate-op apply-op)`y
4. Finally, it returns the *applicability vector* $V$ of every final node $[m; V]$.

```
Procedure applicability-vectors (M_x D)

1. Local definitions
   < node >          ::= [m; V]        ; whith V = [v_1, v_2, ..., v_{s_x}]
   final − node?     ::= λ_1 : < node > ⟶ {#t, #f}
   generate − op     ::= λ_2 : < node > ⟶ (d_1, d_2, ..., d_{s_x})
   apply − op        ::= λ_3 : r × < node > ⟶ [m'; V']
                         with, m' = m − d_r , V' = [v'_1, v'_2, ..., v'_{s_x}],
                         being, v'_r = v_r + 1 but, v'_j = v_j ∀j ≠ r
2. The initial node
   m_0 ← M_x
   V_0 ← [0, 0, ..., 0]
   n_0 ← [m_0; V_0]
3. The final-nodes
   final-nodes ← (breadth-expansion-tree
                  n_0 final-node? generate-op apply-op)
4. Returns the vector V of every node [m; V] of final-nodes
```

*Notes:*

- Every $v_r$ *counts* the times the rule $r$ could be applied.
- $\lambda_1([m; V]) = \#\mathtt{t} \leftrightarrow \forall\, d_r \in D(d_r = \#\mathtt{f} \vee m < d_r)$
- $\lambda_3(r, [m; V]) = \#\mathtt{f}$ if $d_r = \#\mathtt{f}$

### 6.4 The Tag-Rules Procedure

The `tag-rules` procedure *updates* the *app-tag* $\alpha_r$ of those rules $r$ of $R^*$ that are applicable to a given configuration $C = (\mu, M)$. The MzSheme sentence `(tag-rules` $C$ $R^*$ $\rho$`)` returns $R^C$. The procedure works as follows:

1. It starts getting the degree, $N$, of $\Pi$.
2. Then, its work is based on an *external* and an *internal loop*, to go through the membranes and through the rules of every membrane, respectively.
   - The *external loop* generates $R_x^C$, for every $x = 1, 2, \ldots, N$, and, once it is finished, it builds $R^C = (R_1^C, R_2^C, \ldots, R_N^C)$. If $x \notin V(\mu) \vee M_x = \emptyset$, then $R_x^C = R_x^*$, otherwise, $R_x^C$ has to be generated by the *internal loop*.
   - The *internal loop* generates $R_x^C$ for a given $x \in V(\mu)$. It checks the applicability of every rule $r_{x,j} \in R_x^*$ to $C$, it changes $\alpha_{x,j}$ from `#f` to `#t` if so, and it obtains the *tagged-rule* $r_{x,j}^C$; finally, it builds and returns to the *external loop*, $R_x^C = (r_{x,1}^C, r_{x,2}^C, \ldots, r_{x,s_x}^C)$.
3. It returns $R^C$.

```
Procedure tag-rules (C R* ρ)
N ← length of ρ
For every x = 1, 2, ..., N do
If x ∉ V(μ) ∨ M_x = 0 then R_x^C ← R_x^*
else
For every j = 1, 2, ..., s_x do
    If r_{x,j} is not semi-applicable to C then r_{x,j}^C ← r_{x,j}^*
    else
    If ∃k < j | α_{x,k}^C = #t ∧ ρ(k, j) = #t then r_{x,j}^C ← r_{x,j}^*
       else r_{x,j}^C ← (d_{x,j}, v_{x,j}, δ_{x,j}, #t)
    If α_{x,j} = #t then
      For every k < j | α_{x,k}^C = #t ∧ ρ(j, k) = #t do
            α_{x,k}^C ← #f
R_x^C ← (r_{x,1}^C, r_{x,2}^C, ..., r_{x,s_x}^C)
R^C ← (R_1^C, R_2^C, ..., R_N^C)
Return R^C
```

### 6.5 The Apply-Matrix Procedure

The `apply-matrix` procedure computes one *transition step*, $C' = P(C)$, from a configuration, $C = (\mu, M)$, and an *applicability matrix*, $P \in \mathbf{M_{Ap}}(C)$. The MzSheme sentence is `(apply-matrix` $C$ $P$ $R^C$`)`. It works in two steps:

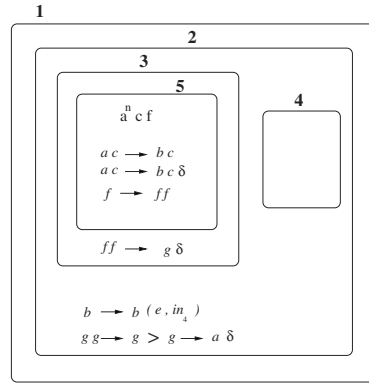
1. For every membrane $x$ in $C$ and every rule $r_{x,j}$ in $R_x^C$, provided $P_{x,j} \neq 0$:
   - $r_{x,j}$ is applied $P_{x,j}$ times *without dissolution*. So, some objects of membrane $x$ are consumed, and maybe itself and/or, its *father* and *children* receive some objects. A more internal loop identifies the *target* where every *action* of the rule sends its objects,
   - then, if $r_{x,j}$ is a dissolution rule, $x$ is stored in $\Delta$ as a node to be dissolved.

2. Then, we visit the nodes of $\mu$ in a *bottom-up ordered* way, the nodes kept on $\Delta$ are dissolved. Every dissolved node sends its objects (and *children*) to its *father* and *disappears* from $\mu$.

```
Procedure apply-matrix (C  P  R^C)
M' ← M
μ' ← μ
Δ ← {}
For every x ∈ V(μ') do
If x ≠ root(μ') then f_x ← the father of x in μ' else f_x ← #f
For every r_{x,j} = (d_{x,j}, v_{x,j}, δ_{x,j}, α_{x,j}) ∈ R_x^C do
    If P_{x,j} ≠ 0 then
     M'_j ← M'_j − P_{x,j} ⊗ d_{x,j}
     For every action = (m, tar) ∈ v_{x,j} do
         If tar ∉ V(μ') then
            If tar = here then tar ← x else tar ← f_x
         If tar ≠ #f then M'_{tar} ← M'_{tar} + P_{x,j} ⊗ m
     If δ_{x,j} = #t then Δ ← Δ ∪ {x}
nodes ← the bottom-up ordered V(μ')
For every x ∈ nodes do
If x ∈ Δ then
   M'_{f_x} ← M'_{f_x} + M'_x
   M'_x ← ∅
   μ' ← delete-node (μ', x)
Return C' = (μ', M')
```

## 7 A Complete Example: Generating Squares $1^2, 2^2, \ldots, n^2$

Finally we present here a complete example to illustrate the way our simulator should be used. The P system to be considered is the following one:



### 7.1 The Input Data

First of all, we have to input the data describing the P system. We do that defining the different elements: $A, N$, *struct*, *output*, *objects*, *rules*, and *orders*. As we need the symbol $a^n cf$, for the given $n$, this one is generated by the auxiliar procedure `generate-symbol`. The MzScheme sentence (`sq1` $n$), assigns the respective value to every compound, and invokes the *parser/compiler*.

```
> (define sq1
   (lambda (n)
    (let ((N 5)
          (A '(a b c e f g))
          (o_m 4)
          (struct '((1 2) (2 3) (2 4) (3 5)))
          (objects
           (vector () () () () (generate-symbol
                   (list 'a n) '(c 1) '(f 1))))
          (rules
           (vector
            '()
            '((b -> ((b here) (e 4)))
              (gg -> ((g here)))
              (g -> ((a here) delete)))
            '((ff -> ((g here) delete)))
            '()
            '((ac -> ((bc here)))
              (ac -> ((bc here) delete))
              (f -> ((ff here))))))
          (orders
           (vector '() '((2 3)) '() '() '())))
      (parser-ps N A struct objects rules orders o_m))))
```

### 7.2 The Parser-Compiler

The *parser/compiler* returns the internal representation of the P system, and displays it in a readable way. So, if $n = 4$ the sentence

```
(define ps (sq1 4))
```

defines, if no error occurs, `ps` as the representation to be used together with the `configurations` procedure.

### 7.3 Configurations

Finally, using the procedure `configurations` to expand the *computation tree*, we obtain *all configurations* until the given level. In particular, with an appropriate level we get all the *final configurations*. In the previous example it is enough to use 9 as depth level.

```
> (configurations ps 9)
TREE:      ((1 2 3 4 5) ((1 2) (2 3) (2 4) (3 5)) 1) ;a non-halting
CONTENTS:                                             ;configuration
  Membrane 1 and Membrane 4:
    Multiset:  #(0 0 0 0 0 0)
    Applic-Rules: #()
  Membrane 2:
    Multiset:  #(0 0 0 0 0 0)
    Applic-Rules: #(#f #f #f)
  Membrane 3:
    Multiset:  #(0 0 0 0 0 0)
    Applic-Rules: #(#f)
  Membrane 5:
    Multiset:  #(0 4 1 0 512 0)
    Applic-Rules: #(#f #f #t)   ;the third rule could be applied
TREE:      ((1 4) ((1 4)) 1)    ;a halting configuration
CONTENTS:
  Membrane 1:
    Multiset:  #(1 4 1 0 0 0)
    Applic-Rules: #()
  Membrane 4:
    Multiset:  #(0 0 0 16 0 0)
    Applic-Rules: #()
TREE:      ((1 4) ((1 4)) 1)   ;a halting configuration
CONTENTS:
  Membrane 1:
    Multiset:  #(2 3 1 0 0 0)
    Applic-Rules: #()
  Membrane 4:
    Multiset:  #(0 0 0 9 0 0)
    Applic-Rules: #()
TREE:      ((1 4) ((1 4)) 1)   ;a halting configuration
CONTENTS:
  Membrane 1:
    Multiset:  #(3 2 1 0 0 0)
    Applic-Rules: #()
  Membrane 4:
    Multiset:  #(0 0 0 4 0 0)
    Applic-Rules: #()
TREE:      ((1 4) ((1 4)) 1)   ;a halting configuration
CONTENTS:
  Membrane 1:
    Multiset:  #(4 1 1 0 0 0)
    Applic-Rules: #()
  Membrane 4:
    Multiset:  #(0 0 0 1 0 0)
    Applic-Rules: #()

Output Membranes:   (() eeeeeeeeeeeeeeee eeeeeeeee eeee e)
```

## 8 Conclusions

Up to now there is no implementation of P systems with a *practical usefulness* that allows the researchers to test and improve the abstract designs they make. The simulation of P systems by conventional programming languages can be considered not only as a practical approach to this computing model, but also as an useful way to understand and improved the P systems designed to solve real problems. We think that, because of the *standard* grammar it uses, the program presented here can be used both as a research tool and a teaching tool, allowing to see the way the P system evolves along its running. The program has been developed in such a way that it could be improved to simulate different variants of P systems. In a future work a graphical interface will be added, to make easier the interaction with the user.

## References

1. Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for Computer Science-TUCS Report* No 208, 1998 (www.tucs.fi).
2. Gh. Păun, G. Rozenberg, A guide to membrane computing, *Theoretical Computer Science*, 287, 1 (2002), 73–100.
3. M.J. Pérez–Jiménez, F. Sancho-Caparrini. A formalization of transition P systems. *Fundamenta Informaticae*, 49, 1-3 (2002), 261–272.
4. M.J. Pérez–Jiménez, F. Sancho-Caparrini. Verifying a *P* system generating squares. *Romanian Journal of Information Science and Technology*, 5, 2–3 (2002), 181–191.
5. MzScheme Home Page. `http://www.cs.rice.edu/CS/PLT/packages/mzscheme/`

# Preliminaries about Some Possible Applications of P Systems in Linguistics

Gemma Bel Enguix

Rovira i Virgili University
Pça Imperial Tárraco, 1, 43005 Tarragona
gbe@astor.urv.es

**Abstract.** Membrane computing [Păun, 2000] is a new and fruitful paradigm of natural computing. The present paper is devoted to some preliminary ideas about how membrane computing can be applied to linguistics. To this end, first some concepts are defined which introduce what we call linguistic P systems. Then, three examples of quite simple applications to pragmatics, phonetic evolution, and dialogue are discussed.

## 1 Introduction

The most important intuition this paper is based on is that membranes can be understood as contexts. Contexts may be different words, persons, social groups, historical periods, languages. They can accept, reject, or produce changes in elements they have inside. At the same time, contexts/membranes and their rules evolve, that is, change, appear, vanish, etc.

This analogy gives rise to a quite suggestive framework to deal with language, understood as an element continuously evolving in a changing world. Language develops and evolves closely related with the environment and, likewise, it can be an active factor of change in its environment. Therefore, following our first analogy, membranes and elements of the system are constantly interacting.

The main elements in the process of communication are the addresser, the addressee, the message, and the context. Syntax is the branch of linguistics which deals with the internal structure of the message. This approach does not seem to be the best to explain how utterances are constructed. Nevertheless, membranes can explain, simulate, and perhaps predict how the elements involved in the communicative process are able to modify the structure or the meaning of the message, and also how the message can create new contexts or transform those which already exist. Pragmatics, language evolution, semantics, sociolinguistics, dialogue, belong to the set of sciences of language that are susceptible to be approached from this perspective.

The present paper is the first attempt to construct a complex game dealing with the simulation of behavior of languages, whose structures, rules, and meanings depend on the space-time coordinates where utterances are generated.

Bearing these goals in mind, linguistic P systems are introduced in Section 2. In Section 3, a preliminary application to pragmatics is discussed. Section 4

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 74–89, 2003.

is devoted to define a P system that is able to reproduce linguistic evolution and, finally, Section 5 is a very preliminary introduction to dialogue in terms of membrane computing.

## 2 Linguistic P Systems

The starting point of linguistic P systems are P systems as they were defined in [Păun, 2000]. Original P systems are designed to generate formal languages, that is, languages that lack interaction with the environment and semantics. In order to adapt P systems to the complexity of verbal language, some concepts must be modified or introduced. In this section, we define some new notions about domains, alphabets, membranes, operations with membranes, as well as new rules. These notions refer to phenomena that occur in human verbal language.

### 2.1 Domains and Alphabets

A linguistic P system has one or more *alphabets*, which can change or evolve during the computation. Each alphabet evolves independently.

The *domain* $\mathcal{D}$ of a membrane is the definition of the symbols it accepts. Domains are related to one or more alphabets (for example, the domain $\mathcal{D}M_n$ can be the union of two alphabets, $V_m \cup V_j$). The domain of the skin membrane is the union of the domains of its internal membranes. Several membranes of the same system can have the same domain.

The function $h$, called function of *emigration*, establishes a correspondence between symbols placed in different membranes. For instance, the function $h(M_n \leftrightarrow M_m)$ establishes the correspondence between symbols belonging to the membrane $M_n$ and symbols belonging to the membrane $M_m$. The rules of this function have the following form: $h(M_n \leftrightarrow M_m) = \{a_{in_{M_n}} \leftrightarrow \alpha_{in_{M_m}}, \ldots, b_{in_{M_n}} \leftrightarrow \beta_{in_{M_m}}\}$. They are called *rules of emigration*. Sometimes, emigration rules have the symbol "$\rightarrow$" instead "$\leftrightarrow$". They are *non-return emigration rules*.

The subscript $_i$ attached to an element in a membrane means that this element is not accepted by the domain of the membrane which it belongs to. If $a$ is not included in the domain of $M_m$ and the rule $h(M_n \leftrightarrow M_m) = a_{in_{M_n}} \leftrightarrow \alpha_{in_{M_m}}$ does not exist, then $a_{in_{M_m}} = a_i$. Elements marked with $_i$ are not taken into account as output of the membrane system when computation stops.

Alphabets can evolve by means of two processes:

1. Some symbols are added to the vocabulary: for example, for $V_1 = \{1, 2, 3\}$, the rule $ADD\ \{4\}\ TO\ V_1$ increases the alphabet so as $V_1 = \{1, 2, 3, 4\}$.
2. Some symbols are erased from the vocabulary: for example, for $V_1 = \{1, 2, 3\}$, the rule $DEL\ \{3\}\ FROM\ V_1$ decreases the alphabet so as $V_1 = \{1, 2\}$.

Domains associated with each membrane can evolve. If they do it, they are called *variable domains*. The processes for changing domains are the following four:

1. New symbols are added to vocabularies belonging to the domain. For example, for $V_1 = \{1, 2, 3\}$ and $\mathcal{D}_{M_n} = V_1$, the rule $ADD\ \{4\}\ TO\ V_1$ increases the domain.
2. Symbols are deleted from vocabularies belonging to the domain. For example, for $V_1 = \{1, 2, 3\}$ and $\mathcal{D}_{M_n} = V_1$, the rule $DEL\ \{3\}\ FROM\ V_1$ decreases the domain.
3. New vocabularies are added to a domain. For example, for $V_1 = \{1, 2, 3\}$, $V_2 = \{a, b, c\}$ and $\mathcal{D}_{M_n} = V_1$, the rule $ADD\ V_2\ TO\ \mathcal{D}_{M_n}$ has as result $\mathcal{D}_{M_n} = V_1 \cup V_2$.
4. A vocabulary is deleted from a domain. $DEL\ V_1\ FROM\ \mathcal{D}_{M_n}$, applied to $\mathcal{D}_{M_n} = V_1 \cup V_2$, has as result $\mathcal{D}_{M_n} = V_2$.

### 2.2 Membranes and Operations with Membranes

We define several operations with the membranes of a linguistic P system.

A *membrane* $M_n$, in a linguistic P system, is defined in each state by means of two items: a) its *domain*, $\mathcal{D}$, and b) the symbols that the membrane contains inside, $(x..z)$. Thus, $M_n = (\mathcal{D}M_n, x..z)$.

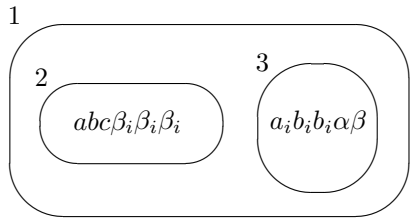



**Fig. 1.** Basic configuration

A linguistic P system can have one or more than one output membranes.

*Deletion* is the operation by means of which a membrane $M_n$ is dissolved and its elements go to the immediately external membrane. These elements will be accepted or rejected according to the definition of the new membrane. The rule for deleting membrane $M_n$ is written as $\delta M_n$.

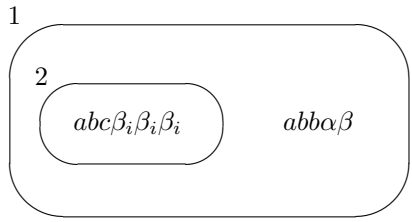



**Fig. 2.** Deletion

*Example 1.* Let $\Pi$ be a P system with three membranes, $[_1\ [_2\ ]_2\ [_3\ ]_3\ ]_1$, where $V_1 = \{a, b, c, d\}$, $V_2 = \{\alpha, \beta\}$, $\mathcal{D}M_2 = V_1$, $\mathcal{D}M_3 = V_2$, $\mathcal{D}M_1 = \mathcal{D}M_2 \cup \mathcal{D}M_3$.

Assume that after some evolution steps the configuration reached is as shown in Fig. 1.

In this moment, the rule $\delta M_3$ is applied, with the result as shown in Fig. 2.

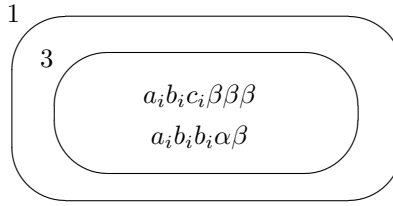



**Fig. 3.** Expansion

*Expansion* is the operation by means of which a membrane $M_n$ can be expanded to other adjacent or external membranes using the rule $\psi M_n$ TO $M_m, M_k$. That means that membranes $M_m$ and $M_k$ are dissolved in $M_n$ and their elements must be reformulated following the definition of $M_n$.

*Example 2.* By applying the rule $\psi M_3$ TO $M_2$ to the system in the previous example, we obtain the situation in Fig. 3.

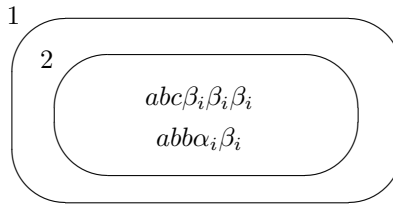



**Fig. 4.** Absorption

*Absorption* is the operation by means of which a membrane $M_n$ disappears dissolved in another adjacent or external membrane $M_m$. Its elements must be reformulated according to the definition of $M_m$. The rule is $\phi M_n \; IN \; _m$.

*Example 3.* If we apply $\phi M_3 \; IN \; M_2$ to $\Pi$, the result is the system in Fig. 4.

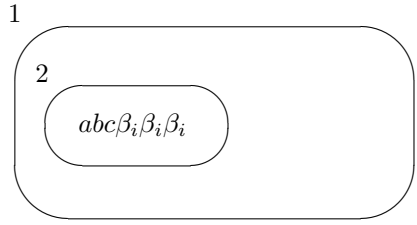



**Fig. 5.** Erasing

*Erasing* is the operation by means of which, given a membrane $M_n$, this can completely disappear with all its elements. The rule is $\chi M_n$

*Example 4.* If we apply $\chi M_3$ to $\Pi$, the result is the system in Fig. 5.

*Cloning* is the operation by means of which a membrane $M_n$, with exactly the same domain and elements, is copied somewhere in the system. The rule has the form $\kappa M_m\ IN\ M_n$.

*Example 5.* For instance, in $\Pi$ we can apply $\kappa M_3\ IN\ M_2$. The result is the system from Fig. 6.

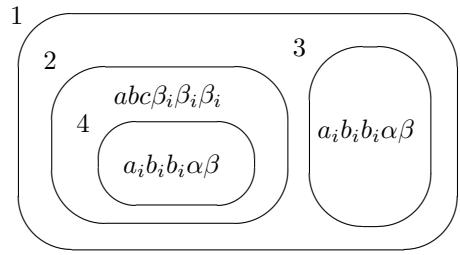



**Fig. 6.** Cloning

In this example, $M_4$ is the clone of $M_3$.

*Invasion* is the operation by means of which a membrane $M_m$ is generated in any place in the system using the rule $\nu M_m\ IN\ M_n$. The new membrane is empty and its domain definition is given by the notation $BY\ M_s$, where $M_s$ is either the membrane where it is generated or the membrane that de rule belongs to.

*Example 6.* For instance, we can crate a new membrane in $M_2$ from the membrane $M_3$. The rule is $\nu M_4\ IN\ M_2\ BY\ M_3$.

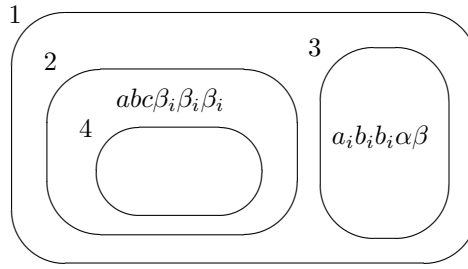



**Fig. 7.** Invasion

In this example $M_4$ has the same domain than $M_3$.

If a linguistic P system has more than one alphabet but only one domain or more than one domain, but only one alphabet, then it is called *non-integrative.*

**Fact 1.** *Let $M_m$ and $M_n$ be two adjacent membranes. Then $\phi M_n\ IN\ M_m = \psi M_m\ TO\ M_n$. Thus, for adjacent membranes, absorption ($\phi$) and expansion ($\psi$) are inverse operations.*

*Example 7.* $\phi M_n \; IN \; M_m = \psi M_m \; TO \; M_n$ (Fig. 8).

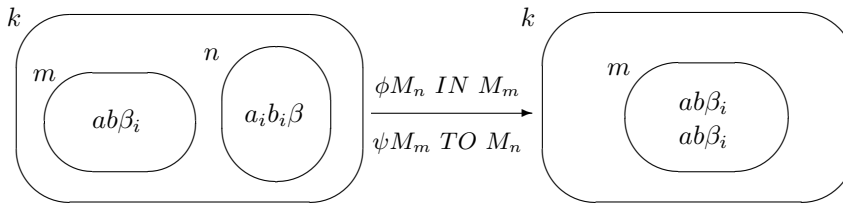



**Fig. 8.** Absorption and expansion as inverse operations

**Fact 2.** *Let $M_m$ be a membrane immediately external to $M_n$. Then $\delta M_n = \phi M_n \; IN \; M_m$.*

*Example 8.* $\delta M_n = \phi M_n \; IN \; M_m$ (Fig. 9).

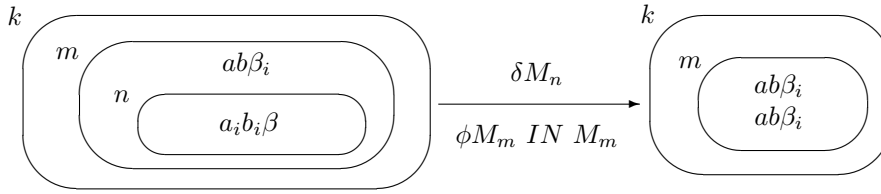



**Fig. 9.** Deletion and absorption as equivalent operations

### 2.3 Inactive and Sleeping Rules

Rules belonging to a membrane can only use the elements that the membrane accepts as domain. If a membrane has any rule with an element that does not belong to its domain, then that rule becomes inactive. If the membrane evolves during the computation and it accepts the necessary element, then the rule is immediately activated.

Sleeping rules are well-formed rules in a given membrane, but they are inactive until some element activates them. Sleeping rules are denoted by $\sigma$. For example, there can exist a rule such that $\sigma r_n : c \to \delta$, but this rule cannot be applied before being activated. There are several ways of activating a sleeping rule (the symbol $\varsigma$ is used to express the ctivation). These rules are the following:

1. To put the sleeping rule into another one which is active. For example, having a sleeping rule $\sigma r_5$, we can writte $r_1 : b \to ab\varsigma r_5 c$.
2. There can exist conditional rules; for instance, consider $IF \; \delta M_3 \to \varsigma r_5$. In this case, if $M_3$ is deleted, then the rule $\sigma r_5$ is activated.

3. Finally, there can be a clock regulating the activation of some sleeping rules. It can be programmed as a counter, so that after some rule or set of rules has been applied several times, another one is activated. For example, a counter $c_x$ can be considered, that increases each time a rule is applied in a system. Then, we can have, for instance, a rule of the form $r_n : c_x = 14 \rightarrow \varsigma r_5$.

The same procedures may be used for deactivating a rule. The only difference is the adjunction of $\sigma$ (the symbol of sleeping) instead of $\varsigma$.

## 3 Applications to Pragmatics

Many definitions of pragmatics have been given. We will deal with several ones that P systems can deal with, those which highlight the connection between linguistic utterances and addressee. In other words, we think that P systems can explain the different interpretation of an utterance depending on the context (personal or social, related to the time or ideology) where it is uttered and decodified. In this sense, some opinions and formalizations are pointed up.

According to [Levinson, 1983, 1], "*the modern usage of the term* pragmatics *is attributable to the philosopher [Morris, 1938], who was concerned to outline (after Locke and Peirce) the general shape of a science of signs, or* semiotics*. Within semiotics, Morris distinguished three distinct branches of inquiry:* syntax*, being the study of 'the formal relation of signs to one another',* semantics*, the study of 'the relations of signs to the objects to which the signs are applicable', and* pragmatics*, the study of 'the relations of signs to the interpreters'".*

As we have seen, according to Morris, syntax, semantics and pragmatics are three constituents of a formal model that is able to explain the systems of meaning. The first and deepest one is syntax, the second one is semantics and, finally, pragmatics is the last component playing in the game of generating utterances with meaning.

Some other authors, as Katz and Fodor, understand pragmatics as the theory concerned with the disambiguation of sentences by the context in which they were uttered. The formal definition is the following [Katz, 1977, 19]:

Let $S$ be the set of sentences in language $L$, $C$ the set of possible contexts, $P$ the set of propositions, and $U$ the cartesian product of $S \times C$, and let the corresponding lower case letters stand for elements or members of each of those sets such as $s \in S$, $c \in C$, $p \in P$, $u \in U$; then

$$f(s, c) = p.$$

Thus, when the meaning of a proposition is established, it is necessary to take into account the utterance plus the context, that is to say, the variations that the context causes in the meaning.

In this way, [Levinson, 1983, 21] uphold that: *Pragmatics is the study of the relations between language and context that are basic to an account of language understanding.*

As a formal recapitulation of all these definitions, we take the Katz' formalization $f(s, c) = p$. We will show the suitability of P systems in the interpretation

of utterances depending on the context. To do so, we need a P system with at least two output membranes, since it is necessary to compare the final elements in each one of them. Rules are thought so as, with only one vocabulary, and the same membrane definition, languages generated in each of the output membranes would be the same. Then, there is a unique utterance, which can change depending on if the addressee accepts or not the message and how it is interpreted by it.

### 3.1 An Example

We can imagine a simple P system with two output membranes, $M_2$ and $M_4$, and generating a regular language in each of them, which are the output membranes. Specifically, let us consider the system:

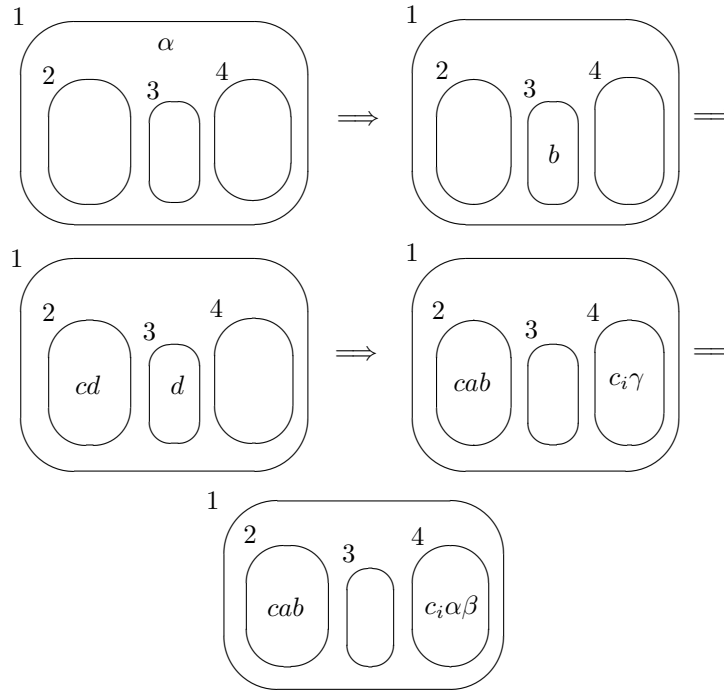



**Fig. 10.** P systems applied to pragmatics

$\Pi = (V_1, V_2, M_1, ..., M_4, \mathcal{D}M_1, .., \mathcal{D}M_4, h(M_2, M_3 \to M_4), h(M_1 \to M_4),$
$(R_1, \rho_1), ..., (R_4, \rho_4), 2, 4),$
$V_1 = \{a, b, c, d\},$
$V_2 = \{\alpha, \beta, \gamma\},$
$\mu = [_1[_2\ ]_2\ [_3\ ]_3\ [_4\ ]_4]_1,$
$M_1 = (V_1 \cup V_2,\ \alpha),$
$M_2 = (V_1,\ \emptyset),$

$M_3 = (V_1, \emptyset)$,
$M_4 = (V_2, \emptyset)$,
$h(M_2, M_3 \to M_4) = (a_{in_{M_{2,3}}} \leftrightarrow \alpha_{in_{M_4}}, b_{in_{M_{2,3}}} \leftrightarrow \beta_{in_{M_4}}, d_{in_{M_{2,3}}} \leftrightarrow \gamma_{in_{M_4}})$,
$h(M_1 \to M_4) = (a_{in_{M_1}} \to \alpha_{in_{M_4}}, b_{in_{M_1}} \to \beta_{in_{M_4}}, d_{in_{M_1}} \to \gamma_{in_{M_4}})$,
$R_1 = \{r_1 : \alpha \to (b, to_3), r_2 : \alpha \to (\chi, to_3)\}$, $\rho_1 = \emptyset$,
$R_2 = \{d \to ab\}$, $\rho_2 = \emptyset$,
$R_3 = \{b \to (cd, to_2)d, d \to (cd, to_4)\}$, $\rho_3 = \emptyset$,
$R_4 = \{\gamma \to (\alpha, to_1)\alpha\beta\}$, $\rho_4 = \emptyset$.
(The target indications $to_j$ are interpreted as "go to membrane $j$", irrespective whether $j$ is an adjacent membrane or not.)

In the first step we can apply the rule $R_1 r_1$ or $R_1 r_2$. With $R_1 r_2$ the system stops immediately, and the outcome is $\emptyset$. Therefore, we will apply $R_1 r_1$. The system evolves as shown in Fig. 10.

In this state, the process can start again or, by means of the application of $R_1 r_2$, $M_3$ disappears and the system stops. In this easy example, by applying the same rules, changing only depending on the domain, two different results are obtained in $M_2$ and $M_4$, the output membranes. When the system stops we have $M_2 = (cab)^n$ and $M_4 = (\alpha\beta)^n$ for $n \geq 0$. Remember that elements as ${c_i}^n$, without rules of emigration, are eliminated when computation finishes.

Therefore, it can be said that, following the formalization given by Katz, $f(s, c) = p$, in this context we have: $f(cab, M_2) = cab$, $f(cab, M_4) = \alpha\beta$.

## 4 Applications to Linguistic Evolution

P systems can be useful for modeling linguistic evolution of languages from a historical point of view. Our intuition is that, by means of the activation/deactivation of rules and the change in alphabets and domains, it is possible to explain the mechanisms of linguistic changes, being phonetic or syntactic ones.

The most important features of linguistic P systems applied to languages evolution are the following:

1. This type of P systems works with ordered sets of elements in each membrane. These ordered sets of elements are strings.
2. In phonetic evolution vowels and consonants play very different roles. Therefore, it seems advisable to establish at least two different alphabets which, if necessary, can act differently. However, the domain of membranes will be composed by vowels and consonants, since phonetic change is carried out according to some laws that combine both types of phonemes.
3. Sometimes, for non-complex systems and quite wide domains, it is possible to work without rules of emigration.
4. Systems of phonetic evolution need some *context sensitive* rules of emigration. Also the rules in each membrane can be context sensitive. Therefore, in this respect, complexity increases.
5. New membranes with associated domains can appear any time during the computation. The process is interpreted as the generation of new tongues.

6. An important feature of rules in linguistic evolution is that they are quite delimited in time and space, that is to say, they act during a "short" time in a local place and later they vanish. Thus, the use of activation/deactivation mechanisms is important.
7. Domains associated to each membrane usually evolve. The most important causes for this change in domains are: a) influence among different contemporary tongues, called *adstrat*, and b) some internal evolutions which modify the set of phonemes existing in the domain of a language.

### 4.1 An Example

In the present example we reproduce three phonetic changes, carried out in different historical moments and languages.

1. *Rotacisme*: $[-s-] > [-r-]$. This change was brought about in IV Century B.C. It causes some strange things like the existence of a nominative *flos* (flower) with a genitive *floris*.
2. $[k^{e,i}-] > [t\int -]$ in vulgar Latin. That process dates from the first centuries of our age. In the example $[t\int]$ is denoted by $c$, for simplicity.
3. $[-t-] > [-d-]$. The alveolar plosive voiceless placed between vowels becomes voiced. This change took place approximately in the VI century A.C. in Western Romance languages.

These changes are simulated in the same membrane without taking into account the chronological order.

The domain in the present P system does not evolve, because it is wide enough to different languages and periods. There are two vocabularies, and the rules of emigration are context sensitive. There is only one output membrane. The P system which can simulate these processes is the following:

$\Pi = (V_1, V_2, M_1, ..., M_4, \mathcal{D}_{M_1}, ..\mathcal{D}_{M_4}, h(\mu \to M_2), (R_1, \rho_1), ..., (R_4, \rho_4), 3),$
$V_1 = \{a, i, u\},$
$V_2 = \{s, k, t, c, d\},$
$\mu = [_1[_2\,]_2\,[_3\,]_3\,[_4\,]_4]_1,$
$M_1 = (V_1 \cup V_2,\ tussa),$
$M_2 = (V_1 \cup V_2,\ \emptyset),$
$M_3 = (V_1 \cup V_2,\ \emptyset),$
$M_4 = (V_1 \cup V_2,\ kk),$
$h(\mu \leftrightarrow M_3) = (ki\{a, u\}_{in_\mu} \leftrightarrow c\{a, u\}_{in_{M_3}},\ {V_1}^+ s {V_1}^+{}_{in_\mu} \leftrightarrow {V_1}^+ r {V_2}^+{}_{in_{M_3}},$
${V_1}^+ t {V_1}^+{}_{in_\mu} \leftrightarrow {V_1}^+ d {V_1}^+{}_{in_{M_3)}},$
$R_1 = \{r_1 : ss \to (k, to_3),\ r_2 : tu \to (a, to_2),\ r_3 : a \to (a, to_3),$
$\rho_1 = \{r_1 > r_2 > r_3\},$
$R_2 = \{r_1 : s \to (s, to_4),\ r_2 : at \to (s, to_4),\ r_3 : t \to (s, to_3)\},$
$\rho_2 = \{r_2 > r_3\},$
$R_3 = \{r_1 : ia \to (t, to_2)t\},$
$R_4 = \{r_1 : k \to (ia, to_3),\ r_2 : s \to (t, to_2)a,\ r_3 : a \to (a, to_3)\},$
$\rho_2 = \{r_2 > r_3\}.$

The system evolves from the initial state as in Fig. 11.

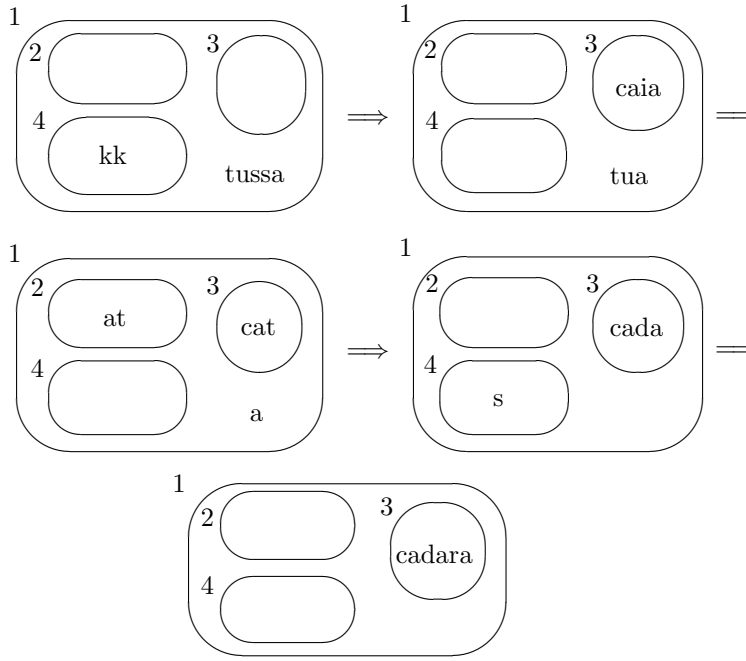



**Fig. 11.** A P system simulating phonetic evolution

The outcome obtained with this system is the same as the one obtained by applying *point mutations*, which allow the change of an element for another one in a string, even depending on the context. The advantage of P systems is that they allow the parallel modeling of different processes. For instance, we can imagine a cellular system to obtain, as an output, two Romance languages, Romanian and Catalan, starting from Latin. The only thing we must do is to consider two output membranes.

Given classical Latin, in order to obtain Catalan, it is necessary to apply some phonetic rules. However, in order to obtain Romanian, some of those rules are not necessary and others must be used.

Let us consider the following phonetic rules:

1. $[k^{e,i}-] > [t\int^{e,i}-]$: a velar sound becomes palatal in vulgar Latin. In the example, $[t\int]$ is denoted by $c$, for simplicity.
2. $[t\int -] >$ [s-] in Catalan.
3. $[-u-] > \emptyset$ in Romanian.
4. $[-u-] >$ [b] in Catalan.
5. Pre tonic vowels disappear, in the whole domain of Romance Languages.

We will define the following linguistic P system without rules of emigration. Membrane $M_1$ does not work because it represents Latin. $M_5$ is a new membrane representing vulgar Latin. Finally, $V_3$ denotes tonic sounds.

$\Pi = (V_1, V_2, V_3, M_1, ..., M_5, \mathcal{D}_1, .., \mathcal{D}_5, (R_1, \rho_1), ..., (R_4, \rho_4), (2, 3))$,
$V_1 = \{i, a, e, u\}$,
$V_2 = \{k, t, s, c, b\}$,
$V_3 = \{T, A\}$,
$\mu = [_1[_2\ ]_2\ [_3\ ]_3\ [_4\ ]_4[_5\ ]_5]_1$,
$M_1 = (V_1\ \cup\ V_2\ \cup\ V_3,\ \emptyset)$,
$M_2 = (V_1\ \cup\ V_2\ \cup\ V_3,\ \emptyset)$,
$M_3 = (V_1\ \cup\ V_2\ \cup\ V_3,\ \emptyset)$,
$M_4 = (V_1\ \cup\ V_2\ \cup\ V_3,\ s)$,
$M_5 = (V_1\ \cup\ V_2\ \cup\ V_3,\ kiuiTAte)$,
$R_1 = \emptyset$, $\rho_1 = \emptyset$,
$R_2 = \{r_1 : viTA \rightarrow TA\}$, $\rho_2 = \emptyset$,
$R_3 = \{r_1 : bT \rightarrow (i, to_4)uT\ \rho_3 = \emptyset$,
$R_4 = \{r_1 : s \rightarrow (si, to_3),\ r_2 : t \rightarrow (b, to_3),\ r_3 : i \rightarrow ATto_3\}$, $\rho_4 = \emptyset$,
$R_5 = \{r_1 : ki \rightarrow ci,\ r_2 : ci \rightarrow (ci, to_2),\ r_3 : ui \rightarrow (ui, to_2)u,\ r_4 : uT \rightarrow (t, to_4)T\}$,
$r_4 : TA \rightarrow (TA, to_2)T$, $r_6 : Tt \rightarrow (T, to_3)$, $r_7 : (e \rightarrow te, to_2)$, $\rho_5 = r_1 > r_2 > r_3 > r_4 > r_5 > r_6 > r_7$.

This system evolves, starting from the initial configuration, as we can see in Fig. 12.

## 5 Applications to Dialogue

Among many existent definitions of dialogue, we choose the one introduced by [Moulin, Rousseau, and Lapalme, 1994, p. 35], who state "*A conversation can be thought of as the result of coordinated interactions among agents to reach a common goal called a conversational goal*". In what concerns the interactions carried out among speakers in a conversation, Levinson [Levinson, 1983, p. 284] says "*Interactions can be understood as the sustained production of chains of mutually-dependent acts, constructed by two or more agents each monitoring and building on the actions of the other.*"

From that point of view, P systems can be constructed that may able to simulate (or generate?) dialogue. Such systems have the following features:

- They do not act in parallel. Despite some rules can be applied at the same time, this will never happen, since the system is blocked (the dialogue is not possible in parallel). By default, the agents are acting in the order $M_1, M_2, .., M_n$, or in a way defined in the system.
- All the membranes (except the *skin membrane*) are output membranes. Even the skin membrane can be an output membrane if it acts as a distributor.
- During the dialogue, some membranes can dissolve or vanish. If only one output membrane remains, then the system stops, for it cannot exist a dialogue with only one output membrane.

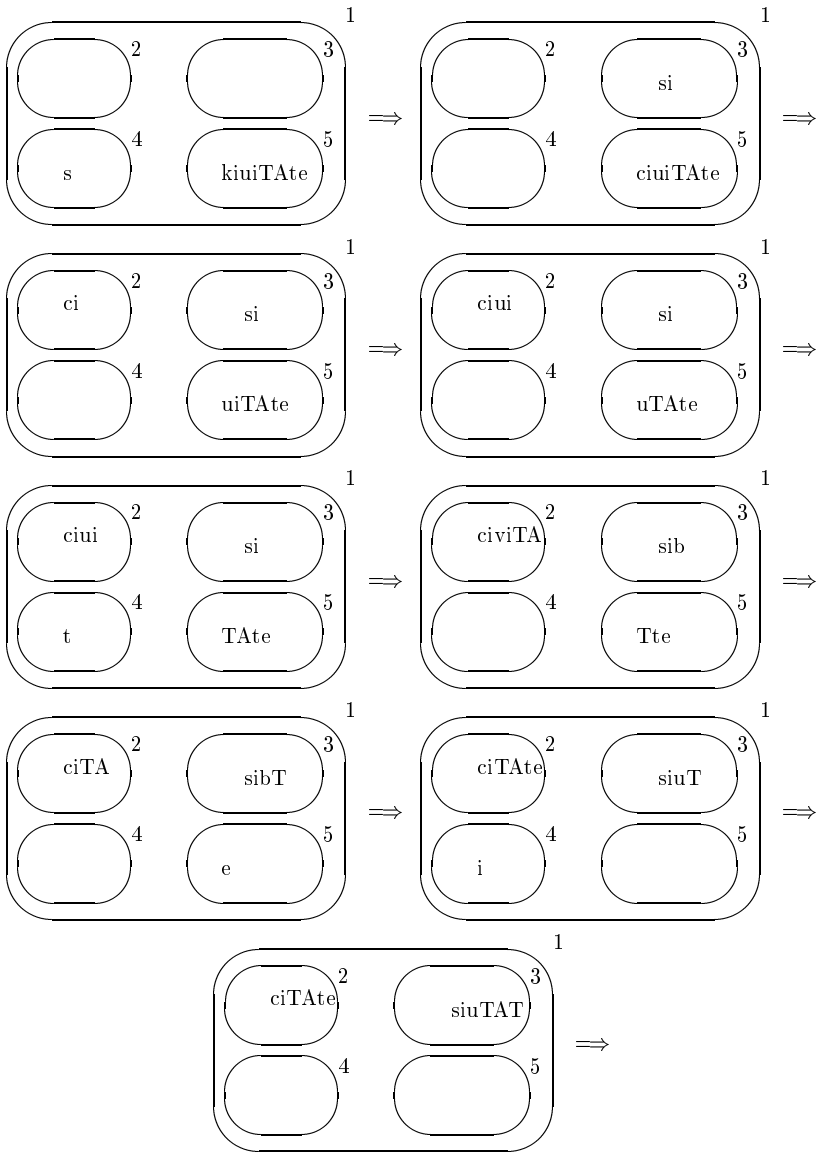



**Fig. 12.** Phonetic evolution: a specific case

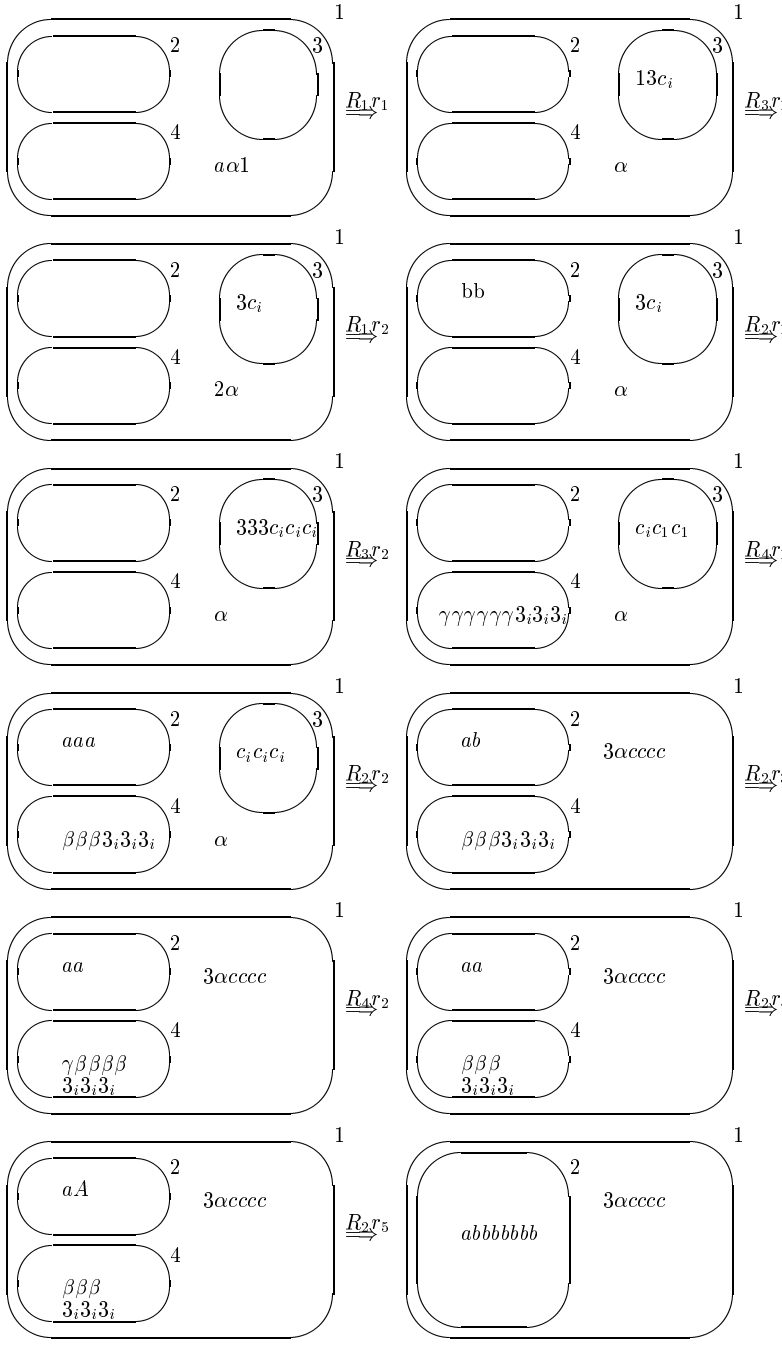



**Fig. 13.** Dialogue: a simple case

Formally, the system is:

$\Pi = (V_1, V_2, V_3, V_4, M_1, ..., M_4, \mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\ h(\mathcal{D}_1 \rightarrow \mathcal{D}_2),\ h(\mathcal{D}_2 \rightarrow \mathcal{D}_3),$
$h(\mathcal{D}_1 \rightarrow \mathcal{D}_3),(R_1, \rho_1), ..., (R_4, \rho_4), (2, 3, 4)),$
$V_1 = \{a, b, c\},$
$V_2 = \{\alpha, \beta, \gamma\},$
$V_3 = \{1, 2, 3\},$
$V_4 = \{A, B, C\},$
$\mu = [_1[_2\ ]_2\ [_3\ ]_3\ [_4\ ]_4]_1,$
$M_1 = (V_1 \cup V_2 \cup V_3,\ 1\alpha a),$
$M_2 = (V_1,\ \emptyset),$
$M_3 = (V_3,\ \emptyset),$
$M_4 = (V_2,\ \emptyset),$
$h(V_1 \leftrightarrow V_2) = (a_{in_{V_1}} \leftrightarrow \alpha_{in_{V_2}},\ b_{in_{V_1}} \leftrightarrow \beta_{in_{V_2}},\ c_{in_{V_1}} \leftrightarrow \gamma_{in_{V_2}}),$
$h(V_1 \leftrightarrow V_3) = (a_{in_{V_1}} \leftrightarrow 1_{in_{V_3}},\ b_{in_{V_1}} \leftrightarrow 3_{in_{V_3}}),$
$h(V_2 \leftrightarrow V_3) = (1_{in_{V_1}} \rightarrow \beta_{in_{V_2}},\ 2_{in_{V_1}} \rightarrow \gamma_{in_{V_2}}),$
$R_1 = \{r_1 : a1 \rightarrow (abc, to_3),\ r_2 : \alpha 2 \rightarrow (bb, to_2)\alpha\},$
$R_2 = \{r_1 : b \rightarrow (bc, to_3),\ r_2 : aa \rightarrow (bc, to_3)b,\ r_3 : b \rightarrow (bc, to_4)a,\ r_4 : aa \rightarrow aA,$
$r_5 : A \rightarrow \psi\ M_2\ TO\ M_3\},$
$R_3 = \{r_1 : 1 \rightarrow (2, to_1),\ r_2 : 3 \rightarrow (223, to_4),\ r_3 : IF|x| \in \mathcal{D}_2 \cup \mathcal{D}_1 \geq 4\ THEN\ \delta\},$
$R_4 = \{r_1 : \gamma\gamma \rightarrow (\alpha, in_2)\beta,\ r_2 : \gamma\beta \rightarrow\ ADD\ V_4\ IN\ \mathcal{D}M_2\}.$

It evolves as shown in Fig. 13. At this point the system stops because there is only one output membrane. In the present dialogue, it can be said that $M_2$ wins.

## 6 Final Remarks

Several suggestions have been given in this paper about applications of membrane computing to linguistics. The paper is only an intuitive and preliminary approach to what seems to be a good way for a computational treatment of some branches of linguistics that usually have strong difficulties for being formalized. Many notions, examples, and applications remain to be considered in this field.

## References

Calude and Păun, 2001. Calude, C. and Păun, Gh. (2001), *Computing with Cells and Atoms*, London, Taylor and Francis.

Katz, 1977. Katz, J.J. (1977), *Propositional Structure and Illocutionary Force*, New York, Crowell.

Katz and Fodor, 1963. Katz, J.J. and Fodor, J.A. (1963), The structure of a semantic theory, *Language*, 39: 170–210.

Levinson, 1983. Levinson, S. (1983), *Pragmatics*, London, Cambridge University Press.

Morris, 1938. Morris, C.W. (1938), Foundations of the Theory of Signs, in Neurath, O., Carnap, R., and Morris, C. (eds.), *Internation Encyclopaedia of Unified Science*, Chichago, University of Chicago Press: 77–138.

Moulin, Rousseau, and Lapalme, 1994. Moulin, B., Rousseau, D., and Lapalme, G. (1994), A multi-agent approach for modelling conversations, in AI'94, *Natural Language Processing. Proceedings of the Fourtheenth International Avignon Conference*, Paris, vol. 3, 35–50.

Păun, 1999. Păun, Gh. (1999), Computing with membranes: An introduction, *Bulletin of the EATCS*, 67: 139–152.

Păun, 2000. Păun, Gh. (2000), Computing with membranes, *J. of Computer and System Sciences*, 61: 108-143.

Tagliavini, 1973. Tagliavini, C. (1973), *Orígenes de las lenguas latinas*, Mexico, Fondo de cultura económica.

# An Application of Dynamic P Systems: Generating Context-Free Languages

Gemma Bel Enguix[1,⋆], Matteo Cavaliere[1,⋆⋆], Rodica Ceterchi[2,⋆⋆⋆], Radu Gramatovici[2], and Carlos Martín-Vide[1]

[1] Research Group on Mathematical Linguistics, Rovira i Virgili University
Pl. Imperial Tarraco 1, 43005 Tarragona, Spain
gbe@astor.urv.es, mc1.doc@estudiants.urv.es, cmv@astor.urv.es
[2] Faculty of Mathematics, University of Bucharest
14, Academiei st., 70109 Bucharest, Romania
{rc,radu}@funinf.math.unibuc.ro

**Abstract.** We present a method of generating context-free languages, in a parallel way, using dynamic P systems, which evolve in time in a coherent manner. The evolution is described by a contextual grammar $D(G)$, which can be canonically associated to any context-free grammar $G$. The dynamic P system generated by $D(G)$ will "compute" the language $L(G)$, i.e., one of the configurations of the system will contain all words of $L(G)$ of length $n$ at depth $2n-1$. Our approach is an attempt to prove the richness and power of the concept of dynamic P system, both in the area of P systems, and in the area of contextual grammars.

## 1 Introduction

We present in this paper a method of generating the words of a context-free language using P systems with string objects and rewriting rules.

The advantage of using P systems lies in the fact that several rewritings can take place, in a parallel manner, each in an appropriate membrane.

The same problem was addressed and solved in [1], but the method proposed here is completely different. Namely, we use a variant of the concept of dynamic P systems, introduced in [2]. However, even if we avoid the use of dynamic P systems, and simulate our method with membrane-generation rules, the two approaches are still different.

We rewrite entire words contained in elementary membranes. After rewriting, the non-terminals still present in the string determine which rules of the initial grammar $G$ are applicable, and a dynamic step generates new elementary

⋆ This author's work has been partially supported by a grant of the Ministerio de Educación y Ciencia GT2001-0017.

⋆⋆ This author's work was possible thanks to a research grant Beca URV from Rovira i Virgili University.

⋆⋆⋆ This author's work was possible thanks to the grant SAB2000-0145, from the Secretaría de Estado de Educación y Universidades, Spanish Ministry for Education, Culture and Sport.

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 90–106, 2003.

membranes with those associated rules, making possible a next step of maximal parallel rewriting. The words of the context-free language $L(G)$ are obtained in elementary membranes.

In section 2 we present some basic notations for context-free grammars and languages, and rewriting membrane systems.

In section 3 we present some notions of contextual grammars that we will use in the sequel of the paper. We define a new type of contextual grammar, the total contextual grammar with an infinite set of contexts, and parallel derivation. It has features already used in the past, and studied in the literature, but not in this combination: total selection function, infinite number of contexts and parallel derivation.

In section 4 we present the notions of dynamic P system with one-step computations, and its associated dynamic computation sequence. While retaining the same general idea, the concepts introduced here are different from those introduced and studied in [2]. The differences concern two essential points: the type (length) of the internal computation step of the dynamic P system in question, and the type of the contextual grammar mechanism used as a generative device. While in [2] dynamic P systems *with stable computations* were studied, here we deal with dynamic P systems *with one-step computations*. Also, the type of contextual grammar used here is not of the same type as that from [2].

In the main section, 5, we make the canonical construction: starting from a context-free grammar $G$, we construct a total contextual grammar $D(G)$ with an infinite number of contexts, and whose associated parallel derivation relation will describe the evolution of the membrane structure. The dynamic P system to which $D(G)$ gives rise, $\Pi(G)$, will "compute" the words of $L(G)$. The main result states that, if $G$ is in Chomsky normal form, then one of the configurations will produce all words of length $n$ at depth $2n-1$. This is similar to the result proved in [1], but the present result is stronger because we produce *all* words in a configuration, while in [1] the "good" configurations produce only one word. Also, if we consider the problem of *collecting* the produced words, it will be easier in the present model, since the words are in elementary membranes, while in the other model a word is represented by its sequence of letters in a nested membrane structure, which makes extraction more difficult.

In section 6 we illustrate the model with an example: we take a context-free grammar $G$ and construct the first terms of the sequence $\Pi(G)$, the dynamic P system canonically associated to it.

Section 7 is devoted to concluding remarks and further research topics. A more detailed comparison with the approach proposed in [1] can also be found there.

## 2 Preliminaries

We recall from [4] the following notations. If $V$ is a non-empty and finite set called *alphabet*, we denote by $V^+$ the set of non-empty words over $V$, with $\lambda$ the empty word and with $V^* = V^+ \cup \{\lambda\}$ the set of words over $V$.

A *context-free grammar* is a construct $G = (N, T, S, P)$ where $N$ and $T$ are alphabets, denoting the sets of *non-terminals* and *terminals*, respectively, $S \in N$ denotes the *start symbol*, and $P \subseteq N \times (N \cup T)^*$ denotes the (finite) set of *production rules*. A production $(A, \alpha) \in P$ will be denoted by $A \to \alpha$. The derivation in $G$ is defined by:

$$x \Longrightarrow y \text{ iff } x = uAv, \;\; y = u\alpha v, \text{ and } A \to \alpha \in P.$$

If $\overset{*}{\Longrightarrow}$ is the reflexive and transitive closure of $\Longrightarrow$, then $L(G) = \{w \in T^* \mid S \overset{*}{\Longrightarrow} w\}$ denotes the language generated by $G$.

We say that a language $L$ is *context-free* if there exists a context-free grammar $G$ such that $L = L(G)$. A context-free grammar $G = (N, T, S, P)$ is in *Chomsky Normal Form (CNF)* iff $P \subseteq N \times (NN \cup T)$. We recall that, for every context-free language $L$, there exists a context-free grammar $G'$ in CNF such that $L(G') = L$.

The grammar from the following example will be used later.

*Example 1. Let us consider* $G = (\{S, A, B\}, \{a, b\}, S, P)$ *a context-free grammar with the productions* ($X \to u|v$ *is a short writing for* $X \to u, X \to v$):

$S \to bA|aB,$
$A \to bAA|aS|a,$
$B \to aBB|bS|b.$

*The generated language is* $L(G) = \{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$, *where* $|w|_\alpha$ *denotes the number of occurrences of the symbol* $\alpha$ *in the string* $w$.

*The context-free grammar* $G' = (\{S, A, B, C_a, C_b, D_1, D_2\}, \{a, b\}, S, P')$ *in CNF is equivalent to the grammar* $G$, *where the productions of* $P'$ *are:*

$S \to C_bA|C_aB,$
$A \to C_aS|C_bD_1|a,$
$B \to C_bS|C_aD_2|b,$
$D_1 \to AA,$
$D_2 \to BB,$
$C_a \to a,$
$C_b \to b.$

*Remark 1.* Using a grammar $G$ in CNF, any word in $L(G)$ of length $n$ is derived in exactly $2n - 1$ derivation steps.

We recall from [11] the following notions.

A rewriting membrane system is a construct of the following form:

$$\Pi = (V, T, \mu, M_1, \cdots, M_m, (R_1, \rho_1), \cdots, (R_m, \rho_m)),$$

where $V$ is an alphabet, $T \subseteq V$ (terminal alphabet), $\mu$ is a membrane structure with $m$ membranes, labelled with $1, 2, \cdots, m$, $M_1, \cdots, M_m$ are finite languages over $V$ (initial string-objects placed in the regions of $\mu$), $R_1, \cdots, R_m$ are finite sets of context-free evolution rules associated with the $m$ regions of $\mu$, and $\rho_1, \cdots, \rho_m$ are partial order relations over $R_1, \cdots, R_m$.

The rules from the sets $R_1, \cdots, R_m$ are of the forms $a \to (v, tar)$ or $a \to (v, tar)\alpha$, where $a \to v$ is a context-free rule over $V$, that is, $a \in V$ and $x \in$

$V^*, tar \in \{here, out, in\}$, and $\alpha \in \{\delta, \tau\}$, with the usual meaning as described in [11]: the string produced by using this rule will go to the membrane indicated by $tar$ ($here$ means that the string remains in the same region, $out$ means to send the string out of the region where the rule is applied, and $in$ means to go to any directly lower membrane, non-deterministically chosen); sometimes one can also use indications of the form $in_j$, where $j$ is the label of the membrane where the string should be sent. To any rule we can append the symbols $\delta, \tau$, indicating that after using the rule the respective membrane changes its permeability as described in [11]. A system with $T = V$ is said to be non-extended, hence a system of the general form is said to be *extended*.

The language generated by $\Pi$ is denoted by $L(\Pi)$ and it is defined as follows: we start from the initial configuration of the system and proceed iteratively, by transition steps performed by applying the rules in parallel, to all strings which can be rewritten, obeying the priority relations. We observe that each string is rewritten by only one rule in a sequential manner and not by rewriting all its symbols. All strings over the alphabet $T$ sent out of the system during any step of any computation form the language $L(\Pi)$.

We denote by $[E]LSP_m(rw, in, pri, \delta, \tau)$ the family of languages generated by [extended] string-object P systems of degree at most $m \geq 1$, using rewriting rules, the target indications $here, out, in$, priorities, and actions $\delta, \tau$. When we use the target indications $in_j$ instead of $in$, we write $LSP_m(rw, tar, pri, \delta, \tau)$; as usual $m$ is replaced with $*$ if the degree of the system is not bounded, and $pri, \delta, \tau$ are removed if the corresponding feature is not used.

In this paper, we use a particular version of a rewriting membrane system: its terminal alphabet $T$ is the terminal alphabet of the grammar to simulate; the rules are of the form $a \to (v, here)$ (hence without $\delta$ or $\tau$, and with $tar$ always fixed as $here$); also, there is exactly one evolution context-free rule associated with each membrane of the system (this means that no priority relation is used). An important point where we depart from the standard notations is the following: in the standard literature, brackets with indices $1, \cdots, m$ are used for the description of a membrane structure $\mu$ as in the definition above, the indices playing the role of correctly matching brackets which express one membrane; in the present paper we have *not* used these kind of indices. The ones we use are the labels of a fixed set of rules, and the expression $[_i \, ]_i$ refers to a membrane having the $i$ rule associated to it.

## 3 Contextual Grammars: Old and New Types

The following notions of contextual grammars and languages are based on definitions from [7]. For the purposes of our model, we will need to extend standard concepts of contextual grammars, more precisely, the concept of total contextual grammar, along two lines, mentioned and used in [7], but only for other types of contextual grammars.

A *total contextual grammar* is a construct $G = (V, A, C, \phi)$, where $V$ is an alphabet, $A$ is a finite language over $V$ (the set of *axioms*), $C$ is a finite subset

of $V^* \times V^*$ (the set of *contexts*), and $\phi : V^* \times V^* \times V^* \to \mathcal{P}(C)$ is the *choice (or selection) map.* The derivation in a total contextual grammar is defined by:

$$x \Longrightarrow y \text{ iff } x = x_1x_2x_3,\ y = x_1ux_2vx_3, \text{ and } (u, v) \in \phi(x_1, x_2, x_3).$$

If $\overset{*}{\Longrightarrow}$ is the reflexive and transitive closure of $\Longrightarrow$, then $L(G) = \{x \in V^* \mid w \overset{*}{\Longrightarrow} x \text{ for some } w \in A\}$ denotes the language generated by $G$. We denote by $TC$ the family of languages generated by total contextual grammars.

Characteristic for total contextual grammars is the highly constrained derivation. As a consequence, total contextual grammars generate languages that cannot be generated by other basic contextual grammars (as external or internal contextual grammars[1]). We mention that the class of context-free languages is entirely included in $TC$. Also, the following three non-context-free constructions – triple agreement, cross dependencies and marked reduplication – which are usually found in natural languages, prove to be of a total contextual type.

*Example 2. Let us consider* $G = (\{a, b, c\}, \{abc\}, \{(a, bc)\}, \phi)$ *a total contextual grammar, where* $\phi$ *is defined by:*

$$\phi(x_1, x_2, x_3) = \begin{cases} \{(a, bc)\} & \text{if } x_1 = a^n, x_2 = b^n, x_3 = c^n, n \geq 1, \\ \emptyset & \text{otherwise.} \end{cases}$$

*We have*

$$L(G) = \{a^nb^nc^n \mid n \geq 1\},$$

*and thus, the language of triple agreement is in* $TC$.

*Example 3. Let us consider* $G = (\{a, b, c, d\}, \{abcd\}, \{(a, c), (b, d)\}, \phi)$ *a total contextual grammar, where* $\phi$ *is defined by:*

$$\phi(x_1, x_2, x_3) = \begin{cases} \{(a, c)\} & \text{if } x_1 \in a^+, x_2 \in b^+c^+, x_3 \in d^+, \\ \{(b, d)\} & \text{if } x_1 \in a^+b^+, x_2 \in c^+, x_3 \in d^+, \\ \emptyset & \text{otherwise.} \end{cases}$$

*We have*

$$L(G) = \{a^nb^mc^nd^m \mid n, m \geq 1\},$$

*and thus, the language of cross dependencies is in* $TC$.

*Example 4. Let us consider* $G = (\{a, b, c\}, \{c\}, \{(a, a), (b, b)\}, \phi)$ *a total contextual grammar, where* $\phi$ *is defined by:*

$$\phi(x_1, x_2, x_3) = \begin{cases} \{(a, a), (b, b)\} & \text{if } x_1 \in \{a, b\}^*, x_2 \in \{c\}\{a, b\}^*, x_3 = \lambda, \\ \emptyset & \text{otherwise.} \end{cases}$$

*We have*

$$L(G) = \{wcw \mid w \in \{a, b\}^*\},$$

*and thus, the language of marked reduplication is in* $TC$.

[1] We do not account here for other constraints on the derivation than the usual selection map.

In the following, we will extend the notion of a total contextual grammar, along two lines. The first extension refers to the cardinality of the set of contexts.

In [7] some contextual grammars with infinite sets of contexts were introduced and studied. More precisely, in section 10.9 of [7], contextual grammars were considered, having an infinite set of contexts $C$, and such that their selection function, $\phi$, has finite image in every point, i.e., $\phi(x)$ is finite for every $x \in V^*$. However, only the classes of languages generated in the internal and the external mode were considered and studied there. For the model which we propose in this paper, a similar notion, allowing for an infinity of contexts, but for total contextual grammars, is needed, so we introduce it in the sequel.

A *total contextual grammar with an infinite set of contexts* is a construct $G = (V, A, C, \phi)$, where all the elements keep their meaning and definition as in a (normal) total contextual grammar, except for the following two modifications:

– the set of contexts, $C$, is not anymore a *finite* subset of $V^* \times V^*$, but an arbitrary one;

– the total selection function is such that $\phi(x_1, x_2, x_3)$ is a *finite* subset of $C$, for any $(x_1, x_2, x_3) \in V^* \times V^* \times V^*$.

Total contextual grammars with an infinite set of contexts generate languages that cannot be generated by (normal) total contextual grammars.

In order to prove this assertion we will use the bounded growth property.

We say that a language $L \subseteq V^*$ has the *bounded growth property* iff there is a bound on the difference between the length of any word in the language and the length of the word immediately longer and belonging to the language, i.e.: there exist two natural numbers $p$ and $q$, such that, for any word $x \in L$ with $|x| > p$, there exists a word $y \in L$ with $0 < |x| - |y| \leq q$.

We know (see [7]) that languages generated by all basic contextual grammars (including the total contextual ones) have the bounded growth property.

Let us denote by $TC_\infty$ the family of languages generated by total contextual grammars with an arbitrary (it may be infinite) set of contexts.

We have the following result.

**Proposition 1.** *The inclusion $TC \subset TC_\infty$ is strict.*

*Proof.* The inclusion itself is trivial. For its strictness, consider the following total contextual grammar with an infinite set of contexts

$$G = (\{a, b, c\}, \{c\}, \{(a, b^{2n+1}c^{2^n}) \mid n \geq 0\}, \phi),$$

where $\phi$ is defined by:

$$\phi(x_1, x_2, x_3) = \begin{cases} \{(a, b^{2n+1}c^{2^n})\} & \text{if } x_1 = a^n, x_2 = b^{n^2}, x_3 = c^{2^n}, n \geq 0, \\ \emptyset & \text{otherwise.} \end{cases}$$

We have

$$L(G) = \{a^n b^{n^2} c^{2^n} \mid n \geq 0\}.$$

$L(G)$ does not have the bounded growth property. If we order the words of this language according to their length, we have

$$L(G) = \{w_0, w_1, \ldots, w_n, \ldots\},$$

with $w_i = a^i b^{i^2} c^{2^i}$, for any $i \geq 0$.

The differences between the lengths of any two consecutive words in this enumeration of $L(G)$ form an infinite sequence of natural numbers $s_n = |w_{n+1}| - |w_n|$, which converges to $\infty$. Therefore, regardless of the choice of the constant $p$ from the definition of the bounded growth property, there exists no constant $q$ which may bound the difference between a word $x \in L(G)$ longer than $p$ and *any* other word in $L(G)$, shorter than $x$. □

The second extension of the notion of a total contextual grammar concerns the parallelism of derivations.

A notion of parallel derivation is defined in [7] (section 10.5), for (classical) internal contextual grammars (with a finite set of contexts).

We generalize this notion of parallel derivation for total contextual grammars. While in the case of internal contextual grammars the string to which the derivation is applied is decomposed into selectors, in our case the string will be decomposed into *triple selectors*.

The *parallel derivation* in a total contextual grammar (with or without an infinite set of contexts) $G = (V, A, C, \phi)$ is defined by:

$$\begin{aligned} x \Longrightarrow_p y \text{ iff } & x = x_1 x_2 \ldots x_n \text{ and } y = y_1 y_2 \ldots y_n, \text{ where} \\ & x_i = x_{1,i} x_{2,i} x_{3,i} \text{ and } y_i = x_{1,i} u_i x_{2,i} v_i x_{3,i}, \text{ with} \\ & (u_i, v_i) \in \phi(x_{1,i}, x_{2,i}, x_{3,i}), 1 \leq i \leq n, \ n \geq 1. \end{aligned}$$

The two newly added features – the infinity of the set of contexts, and the parallelism of the derivation – can be combined, giving rise to *total contextual grammars with an infinite set of contexts and parallel derivation*, and in section 5 we will construct precisely such a grammar.

## 4 Dynamic P Systems

The notion of dynamic P system was first introduced in [2].

In the most general sense, a dynamic P system is a P system which changes/evolves in time, in a coherent manner (aside from the computations done inside it), and the changes are made via a contextual grammar mechanism. We can also see the contextual grammar mechanism as *describing* an evolutionary process intrinsic to the system.

The idea to use a grammar to describe changes in a system, more specifically in a P system, is not so unexpected; but it requires a "good" string-description of a type of P systems. The derivation associated to the grammar takes us from one system to another one, in a coherent way.

More specifically, suppose we have a set of well-formed strings (well-formed according to some formal definition), denoted $Exp(V)$, over an alphabet $V$, and containing also separators. Among the separators we will use the brackets $\{[,]\}$, to describe the membrane structure of the system, but we can use also other kinds of separators (like | and ; in [2]). The strings in $Exp(V)$ describe some particular type of P systems.

Suppose that we also have a contextual grammar $D$ of a certain type (internal contextual, total contextual, insertion grammar, etc.) with the following properties:

(1) its set of axioms is contained in $Exp(V)$;

(2) its derivation relation, $\Longrightarrow_D$, keeps us inside the set of well-formed strings, i.e., if $x \in Exp(V)$ and $x \Longrightarrow_D y$, then $y \in Exp(V)$.

Then, starting from a P system described by an axiom of $D$, and applying repeatedly derivations of $D$, we obtain a sequence of P systems which have "evolved" in a coherent manner from the original one. We can also conceive of this mechanism as being not a generative device, but a descriptive tool.

We can use contextual grammars and their associated derivation to describe any kind of changes in a P system – changes in string or object contents, or rule contents, but, of course, the most interesting ones are changes of the membrane structure, maybe accompanied by other changes as well.

On the other hand, in a P system we have "internal computations", which can be one or several of the following operations: string rewriting, communication by means of symport/antiport rules, moving symbol objects, etc. (see [11] and [12] for a longer list of operations). The computations inside a P system take place in general in a maximal parallel way. We will call *one computation step* what is known in the standard literature of P systems as *a transition*: the passage from one configuration of the P system to the next one.

If $\Pi$ is a P system, let us denote by $\Pi \rightsquigarrow C\Pi$ one internal computation step, the new P system $C\Pi$ being the result of the computation step.

**Definition 1.** *A* dynamic P system with one-step computations, associated to the grammar $D$ *will be a sequence of P systems* $\{\Pi_n \mid n \geq 1\}$ *such that:*

*(i)* $\Pi_1$ *is an axiom of* $D$*;*

*(ii) for each* $i \geq 1$, $C\Pi_i$ *denotes the P system obtained from* $\Pi_i$ *after one-step computations;*

*(ii) for* $i \geq 2$, *each* $\Pi_i$ *is obtained from* $\Pi_{i-1}$ *by one derivation in the grammar* $G$*, that is* $C\Pi_{i-1} \Longrightarrow_D \Pi_i$.

*The* dynamic computation sequence *associated to the above system is the following alternating sequence of derivations in* $G$ *and one-step computations:*

$$\Pi_1 \rightsquigarrow C\Pi_1 \Longrightarrow_D \Pi_2 \rightsquigarrow C\Pi_2 \Longrightarrow_D \ldots \Longrightarrow_D \Pi_i \rightsquigarrow C\Pi_i \Longrightarrow_D \ldots .$$

The dynamic P systems considered in [2] were of a slightly different type, namely, they were dynamic P systems *with stable computation steps* (see [3]), in which the internal computations were allowed to take place till the system reached a stable configuration.

A dynamic computation sequence might begin either with an internal computation, or, if this is not possible, i.e., $\Pi_1 = C\Pi_1$, with a derivation in $G$.

We say that a dynamic P system is *stationary* if the sequence $\{\Pi_n \mid n \geq 1\}$ is stationary in the usual sense, i.e., there exists an index $m$ such that for all $k \geq m$ $\Pi_k = \Pi_{k+1}$. This means that neither internal computations, nor derivations in $D$ are possible in $\Pi_m$ or $C\Pi_m$.

In [2] and [3] we have dealt with P systems with symport/antiport rules, first without string-objects inside, then allowing string-objects in some of the membranes, and finally, in the most general form, allowing string-objects inside any of the membranes. In the present paper we will deal with completely different types of P systems: they contain string objects only in the elementary membranes, and each membrane has one and only one rewriting rule associated with it. Each membrane is *typed* according to the unique string rewriting rule it contains. The indices of the brackets representing membranes will refer, in this paper, to types, and not to some indexing method generally used to eliminate ambiguity by matching appropriate pairs of open and closed brackets, like they are used in general.

The contextual dynamic mechanism presented in [2] and [3] is based on the notion of enriched bracketed contextual grammar, a generalization of the bracketed contextual grammars of [6]. Even if we still use brackets to delimitate membranes, the total contextual grammar with an infinite set of contexts $D(G)$ which we will construct in the next section, in order to describe a different dynamic of a different type of P system, is *not* essentially of a contextual bracketed type.

P systems which allow for changes in the membrane structure (generation of new membranes, merging, dissolving, etc.) have been considered in the literature on P systems (see [11], [12], [5]). The main difference between other approaches and ours is that other approaches describe membrane behavior by rewriting rules, for instance, placed at the same level as the object or string handling rules, while in our approach the changes are generated/described by a contextual grammar mechanism (that is way we call them "coherent"), and the mechanism works in an alternating pattern with the rest of the internal operations.

## 5 The Contextual Dynamic Mechanism

In what follows we will consider a fixed context-free grammar, $G = (N, T, S, P)$, with the rules in $P$ labelled by integers from 1 to $n$. We can suppose that $G$ is in CNF, but for the time being this is not essential.

Let $V = N \cup T$, and consider also the alphabet of separators $Sep = Sep_{[} \cup Sep_{]}$, where $Sep_{[} = \{[_i \mid 0 \leq i \leq n\}$, $Sep_{]} = \{]_i \mid 0 \leq i \leq n\}$, i.e., the set of pairs of open and closed brackets indexed by the rules in $P$, together with an extra pair of brackets indexed by 0. We will use the separators to denote membranes with associated rules from $P$, for instance $[_i]_i$ will denote an empty membrane having associated the rule $i$ of $P$.

**Definition 2.** *A string* $e \in (V \cup Sep)^*$ *will be called* a G-expression *iff:*

*(i) either* $e = [_i \alpha]_i$, *with* $\alpha \in V^+$, *or*
*(ii)* $e = [_i e_1 \dots e_m]_i$, *with* $e_1, \dots, e_m$ *G-expressions.*

Denote by $E(G)$ the set of all G-expressions. Consider on $E(G)$ the equivalence relation defined by:

$$[_i e_1 \dots e_m]_i \sim [_i e_{\sigma(1)} \dots e_{\sigma(m)}]_i,$$

for any permutation $\sigma$ of $m$ elements.

The equivalence class of a string $e$ in $E(G)$, denoted still by $e$, will describe a membrane structure with the properties:

- every membrane, with the exception of the skin membrane $[_0]_0$, has an associated rule in $P$ (i.e., every membrane is typed according to this unique rule, and $[_i]_i$ is the notation for membrane with rule $i$),
- every elementary membrane contains a string from $V^+$, and no other membrane contains strings.

Let us consider now the total contextual grammar with an infinite set of contexts, canonically associated to the fixed context-free grammar $G$:

$$D(G) = (V \cup Sep, \{[_0 S]_0\}, C, \phi),$$

where $C$ is defined by:

$$C = \{([_{i_1}, ]_{i_1}[_{i_2}\alpha]_{i_2} \dots [_{i_k}\alpha]_{i_k}) \mid \alpha \in V^+ \setminus T^+, 1 \le i_j \le n, 1 \le j \le k, k \ge 1,$$
$$\text{where } \{i_1 < \dots < i_k\} \text{ is the ordered set of all labels of all rules in } P$$
$$\text{which have as left-hand side the non-terminal symbols of } \alpha\} \cup \{(\lambda, \lambda)\}$$

and $\phi$ is defined by:

$$\phi(x_1, x_2, x_3) = \begin{cases} \{([_{i_1}, ]_{i_1}[_{i_2}\alpha]_{i_2} \dots [_{i_k}\alpha]_{i_k})\} & \text{if } x_1 = \beta[_i, x_2 = \alpha \in V^+ \setminus T^+, \\ & x_3 = ]_i\gamma, 1 \le i \le n, \\ & \beta \in Sep_{[}^*, \gamma \in Sep_{]}^*, \\ \{(\lambda, \lambda)\} & \text{if } x_1 = \beta[_i, x_2 = \alpha \in T^+, x_3 = ]_i\gamma, \\ & 1 \le i \le n, \beta \in Sep_{[}^*, \gamma \in Sep_{]}^*, \\ \emptyset & \text{otherwise.} \end{cases}$$

Note that $[_0 S]_0 \in E(G)$.

Note also that, if $\alpha \in V^+ \setminus T^+$, then $\{i_1 < \dots < i_k\}$ is the ordered set of all possible rules in $P$ which can be used to rewrite $\alpha$ in $G$.

If $\alpha \in T^*$, then actually no insertion of contexts takes place in $D(G)$, so we can replace this with "no rule" (the power of the empty context).

For $\alpha \in V^+ \setminus T^+$, $\beta \in Sep_{[}^*, \gamma \in Sep_{]}^*$, the (normal) derivation relation $\Longrightarrow$ in $D(G)$ is:

$$\beta[_i \alpha]_i \gamma \Longrightarrow \beta[_i[_{i_1}\alpha]_{i_1} \dots [_{i_k}\alpha]_{i_k}]_i\gamma.$$

In terms of membrane structures, the application of the above derivation has the effect of transforming the elementary membrane $i$ which contains the string $\alpha$, in membrane $i$, no longer elementary, containing submembranes $i_1, i_2, \ldots, i_k$, each of them in possession of a copy of $\alpha$.

In terms of P systems, this derivation works like a rule of "membrane creation with string replication":

$$[_i \alpha]_i \rightarrow [_i [_{i_1} \alpha]_{i_1} \ldots [_{i_k} \alpha]_{i_k}]_i,$$

where the types of the new generated membranes depend essentially on the string content $\alpha$ of the initial membrane $i$.

We have the following, easy to prove, result:

**Lemma 1.** *If $x \in E(G)$ and $x \Longrightarrow y$, then $y \in E(G)$.*

The parallel derivation relation in $D(G)$ will transform in the above manner *all* elementary membranes whose strings contain nonterminals.

**Lemma 2.** *If $x \in E(G)$ and $x \Longrightarrow_p y$, then $y \in E(G)$.*

From the above lemmas, it also follows:

**Lemma 3.** *1. If $x \in E(G)$ and $x \overset{*}{\Longrightarrow} y$, then $y \in E(G)$.*
*2. If $x \in E(G)$ and $x \overset{*}{\Longrightarrow}_p y$, then $y \in E(G)$.*

Note that, from the way the derivation in $D(G)$ was defined, the most recently generated elementary membranes can perform internal computations, in the form of rewriting, on the string they contain. Actually, the derivations in $D(G)$ generate *only* such membranes. Because of this reason, the grammar $D(G)$ generates a dynamic P system $\Pi(G) = \{\Pi_n \mid n \geq 1\}$, with the following associated dynamic computation sequence:

$$\Pi_1 = [_0 S]_0 = C\Pi_1 \Longrightarrow_p \Pi_2 \rightsquigarrow C\Pi_2 \Longrightarrow_p \ldots \Longrightarrow_p \Pi_n \rightsquigarrow C\Pi_n \Longrightarrow_p \ldots$$

Each derivation step will change the *membrane structure*; actually, it will enrich it by adding new elementary membranes, and increasing the depth of the membrane structure by 1. Each internal computational step will change *the string content* of the elementary membranes, by performing a rewriting operation.

The processes are strongly linked together: the derivation step creates the conditions for an internal computational step to take place, and the result of the internal rewriting determines the next derivation, the next change in the membrane structure.

There are several possible *configurations* of the above dynamic P system, which arise from the nondeterminism of the one-step internal computations, i.e., the rewriting process: if a non-terminal occurs more than once in a string $\alpha$, then there are several rewritings of $\alpha$, using the same rule. This will influence both the possible next derivation step, and also the next internal computation step, that is, the next rewriting.

We have the following theorem, the main result of our paper:

**Theorem 1.** *Let $G$ be a context-free grammar, and $D(G)$ its canonically associated grammar, i.e., the total contextual grammar with an infinite number of contexts and having parallel derivations, as constructed above.*

*Let $\Pi(G) = \{\Pi_n \mid n \geq 1\}$ denote the dynamic P system with one-step computations associated to $D(G)$.*

*The following assertions are true:*

*(i) For all the terms of the dynamic P system $\Pi(G)$, all the strings inside their membranes are sentential forms of the initial grammar $G$.*

*(ii) If the language $L(G)$ is finite, then the dynamic P system $\Pi(G)$ is stationary.*

*(iii) If the grammar $G$ is in Chomsky normal form, then there exists one configuration which, at depths $2n-1$, contains all words of length $n$, i.e., the set $\{w \in L(G) \mid |w| = n\}$.*

*Proof.* For assertion (i): it is true of the first term of the sequence. The terms of type $C\Pi_n$ are the results of a one-step internal computation, which consists precisely of one application of a derivation in $G$ in the membranes where this is possible. The terms of type $\Pi_n$ result from a derivation in the grammar $D(G)$, and the derivations do not alter the string contents of the membranes, they just create "optimal" conditions (membranes) for other derivations in $G$ to take place.

For (ii) use the fact that, once a word in $L(G)$ is produced in a membrane, then, in the corresponding string description of the entire P system, we have no more selectors for applying a derivation which adds contexts others than $(\lambda, \lambda)$.

For (iii) use remark 1. □

The assertion (*iii*) of our theorem resembles very much, in spirit if not in form, Theorem 1 of [1].

## 6 An Example

We illustrate our generative mechanism in this section, with the example of a dynamic P system with one-step computations, and its associated dynamic computation sequence, associated to a CF grammar.

We use the grammar in Example 1, $G = (\{S, A, B\}, \{a, b\}, S, P)$ with the set of productions, $P$, which we list below, and label each production with an integer:

(1) $S \to aB$, (3) $A \to bAA$, (6) $B \to aBB$,
(2) $S \to bA$, (4) $A \to aS$, (7) $B \to bS$,
(5) $A \to a$, (8) $B \to b$.

Each membrane will have an associated rule in $P$, and will be labelled with the corresponding integer. The skin membrane does not have any rule associated to it.

Let us consider the P system $\Pi_1 = [_0\ S\ ]_0$, and take it as the axiom of our dynamic grammar $D(G)$. $\Pi_1$ consists of the skin membrane (with no rules),

in which we have fed the start symbol $S$. There are no internal computations possible inside this system, so $\Pi_1 = C\Pi_1$.

Next follows a dynamic step, a derivation in $D(G)$, with the following general description: for every derivation which can be applied to a string inside a membrane, we generate a sub-membrane associated to that derivation, and feed a copy of the string inside. In our case, there are two rules which can be applied, labelled (1) and (2). We have

$$\phi([_0\ ,\ S\ ,\ ]_0) = \{([_1, ]_1[_2S]_2)\},$$

and the derivation $\Pi_1 \Longrightarrow_p \Pi_2$ will generate precisely $\Pi_2 = [_0[_1S]_1[_2S]_2]_0$.

Now, an internal computation step is possible, since rules (1) and (2) can, in their respective membranes, rewrite $S$. The one-step internal computation $\Pi_2 \rightsquigarrow C\Pi_2$ leads to the P system $C\Pi_2 = [_0[_1aB]_1[_2bA]_2]_0$. The transition is depicted in Figure 1.

Since

$$\phi([_0[_1, aB, ]_1) = \{([_6, ]_6[_7aB]_7[_8aB]_8)\},$$
$$\phi([_2, bA, ]_2]_0) = \{([_3, ]_3[_4bA]_4[_5bA]_5)\},$$

the parallel derivation step $C\Pi_2 \Longrightarrow_p \Pi_3$ will give us:

$$\Pi_3 = [_0[_1[_6aB]_6[_7aB]_7[_8aB]_8]_1[_2[_3bA]_3[_4bA]_4[_5bA]_5]_2]_0.$$

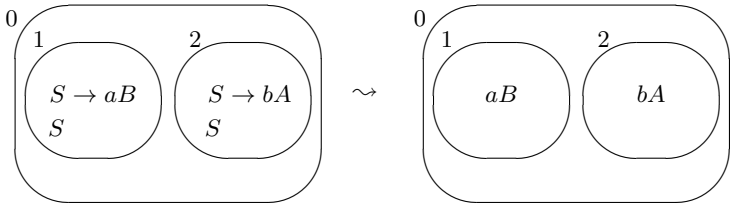



**Fig. 1.** First internal computation step: $\Pi_2 \rightsquigarrow C\Pi_2$

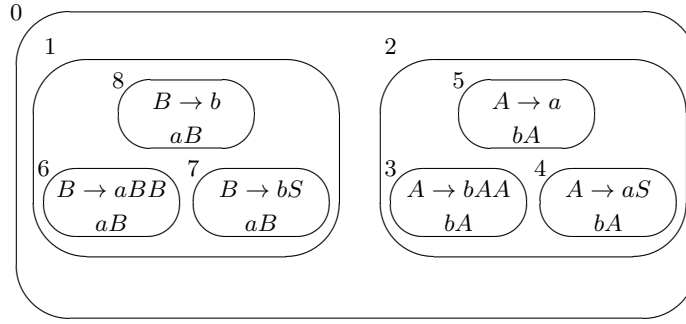



**Fig. 2.** The system $\Pi_3$

The second internal computation, $\Pi_3 \rightsquigarrow C\Pi_3$ will produce:

$$C\Pi_3 = [_0[_1[_6aaBB]_6[_7abS]_7[_8ab]_8]_1[_2[_3bbAA]_3[_4baS]_4[_5ba]_5]_2]_0,$$

and is depicted in Figure 3.

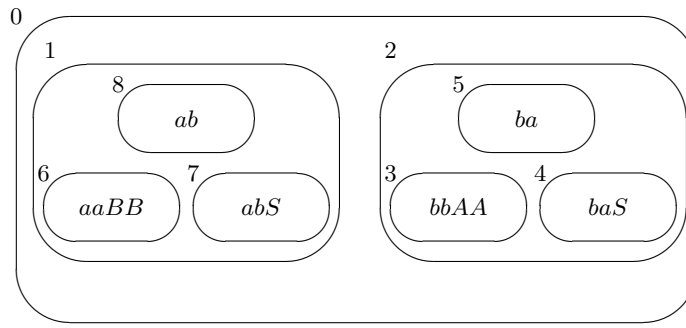



**Fig. 3.** Second internal computation step: $\Pi_3 \rightsquigarrow C\Pi_3$

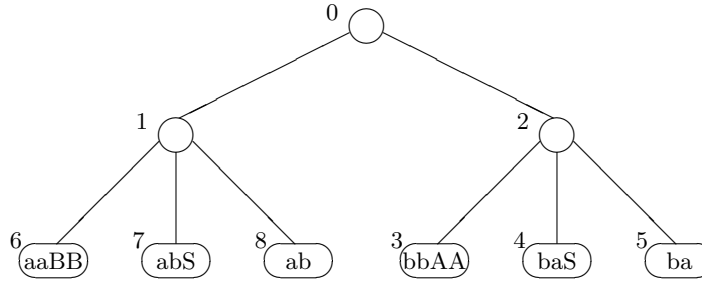



**Fig. 4.** The tree structure of $C\Pi_3$

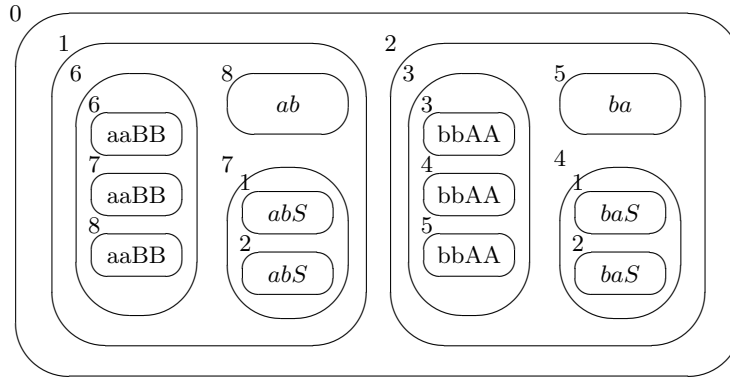



**Fig. 5.** $\Pi_4$, the result of the third derivation step, $C\Pi_3 \Longrightarrow_p \Pi_4$

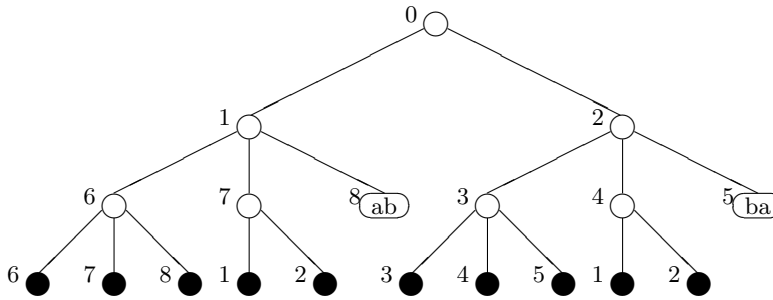



**Fig. 6.** The tree structure of $\Pi_4$. The black nodes are the elementary membranes where rewriting will take place during the next internal computation step. Note that the nodes whose string content is in $T^*$ are leaves in this tree, and will remain leaves in subsequent trees

Since we have the following values for the selection function:

$\phi([_0[_1[_6, aaBB, ]_6) = \{([_6, ]_6[_7 aaBB]_7[_8 aaBB]_8)\}$,
$\phi([_7, abS, ]_7) = \{([_1, ]_1[_2 abS]_2\}$,
$\phi([_2[_3, bbAA, ]_3) = \{([_3, ]_3[_4 bbAA]_4[_5 bbAA]_5)\}$,
$\phi([_4, baS, ]_4) = \{([_1, ]_1[_2 baS]_2)\}$,
$\phi([_8, ab, ]_8]_1) = \{(\lambda, \lambda)\}$,
$\phi([_5, ba, ]_5]_2]_0) = \{(\lambda, \lambda)\}$,

the total derivations in $D(G)$ which will be applied in parallel to $C\Pi_3$ are:

$[_0[_1[_6 aaBB]_6 \Longrightarrow [_0[_1[_6[_6 aaBB]_6[_7 aaBB]_7[_8 aaBB]_8]_6$,
$[_7 abS]_7 \Longrightarrow [_7[_1 abS]_1[_2 abS]_2]_7$,
$[_2[_3 bbAA]_3 \Longrightarrow [_2[_3[_3 bbAA]_3[_4 bbAA]_4[_5 bbAA]_5]_3$,
$[_4 baS]_4 \Longrightarrow [_4[_1 baS]_1[_2 baS]_2]_4$,
$[_8 ab]_8]_1$, $[_5 ba]_5]_2]_0$ no actual derivations take place.

Finally, we will have $C\Pi_3 \Longrightarrow_p \Pi_4$, with

$$\Pi_4 = [_0[_1[_6[_6 aaBB]_6[_7 aaBB]_7[_8 aaBB]_8]_6[_7[_1 abS]_1[_2 abS]_2]_7[_8 ab]_8]_1 \\ [_2[_3[_3 bbAA]_3[_4 bbAA]_4[_5 bbAA]_5]_3[_4[_1 baS]_1[_2 baS]_2]_4[_5 ba]_5]_2]_0.$$

## 7 Concluding Remarks

We have proposed in this paper a generative mechanism for context-free languages, which uses parallelism, and the parallelism is implemented using the notion of a P system. As such, it is in some ways similar to the approach in [1].

It is interesting to note the differences and the similarities of the two approaches. The central idea of [1] is to use evolution rules for symbol-objects, and simulate the derivations of a context-free grammar using membrane operations (division of neutral polarized membranes, subordination of membranes polarized positively and negatively, see [1]). In this way, all strings of $L(G)$ are generated, composing sequences of symbol-objects and storing the order of the derived sentential forms in the tree-structure of the membrane system. In this

way, *every single derivation is made in a parallel manner*, but the system can execute only one derivation at a time – so the totality of strings is derived using the non-deterministic behavior of the system.

In our case, we use string-objects to store the sentential forms, and we use rewriting rules for the simulation of the productions in $G$. So, every single derivation is realized in a sequential way, but, on the other hand, we are able to *generate many derivations in parallel*, using the intrinsic parallelism of P systems. Compared with the standard sequential derivation in the context-free grammar $G$, our approach using P systems allows for the possibility of having different derivations taking place simultaneously, in a parallel way.

We also avoid "garbage" generation (i.e., generation of membranes unable to perform internal computations), by using the *dynamic* P systems. Therefore the degree of non-determinism of our dynamic P system (a non-determinism which arises only from the "P system behavior" of the dynamic computation sequence) is less than that in [1].

Our approach can be easily generalized to attack the problem of generating the language defined by a context-sensitive or a type-0 grammar. Our idea is to give a "framework" based on dynamic P systems, where it should be possible to generate, in a "coherent" way (and in a parallel way) the language defined by such a grammar. We know that, usually, in the P system area, the simulation of a grammar with a P system does not pay too much attention to aspects such as avoiding "garbage" generation, and achieving low degrees of non-determinism. We have seen that with dynamic P systems we can deal with these problems.

As special case of the more general question, an interesting (open) problem can be proposed in this framework: find a P system which, using rewriting rules and creation of membranes, can generate, in a totally deterministic way, all the words of a context-free language. We can also observe that such a problem is related with the study of a deterministic parallel parsing algorithm for context-free languages.

We hope also to have proved that the new concept of *dynamic P system*, a system whose *coherent evolution* in time is described by a *contextual grammar* mechanism, is an elegant tool which can be used in different settings to solve problems using P systems.

The rather inedited type of contextual grammar which we used in this paper to control the evolution of the dynamic P system, the total contextual grammar with an infinite set of contexts, working in the parallel derivation style, also deserves further investigation.

## References

1. A. Atanasiu, C. Martin-Vide: *P Systems and Context-Free Languages*, in Actas del Primer Congresso Español de Algoritmos Evolutivos y Bioinspirados, AEB'02, Mérida, Febrero, 2002, 341–346
2. R. Ceterchi, C. Martin-Vide: *Generating P Systems with Contextual Grammars*, Pre-proceedings of the "Workshop on Membrane Computing", Curtea de Argeş, 2002

3. R. Ceterchi, C. Martin-Vide: *Dynamic P Systems*, this volume
4. J.E. Hopcroft, J.D. Ullman: *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading MA, 1979
5. M. Margenstern, C. Martin-Vide, Gh. Păun: *Computing with Membranes: Variants with Enhanced Membrane Handling*, Proc. of 7th Intern. Meeting on DNA Based Computers (N.Jonoska, N.C.Seeman ed.), Tampa, Florida, USA, 2001, 53–62
6. C. Martin-Vide, Gh. Păun: *Structured Contextual Grammars*, Grammars, 1, 1 (1998), 33–55
7. Gh. Păun: *Marcus Contextual Grammars*, Kluwer, Dordrecht, 1997
8. Gh. Păun: *Computing with Membranes. An Introduction*, Bulletin of EATCS, 67 (1999), 139–152
9. Gh. Păun: *Computing with Membranes*, Journal of Computer and System Sciences, 61, 1, (2000), 108-143, and Turku Center for CS – TUCS Report No.208, 1998
10. Gh. Păun: *P Systems with Active Membranes: Attacking NP-Complete Problems*, Journal of Automata, Languages and Combinatorics, 6, 1 (2001),75–90
11. Gh. Păun: *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, Heidelberg, 2002
12. *http://psystems.disco.unimib.it/*

# P Systems with Boundary Rules

Francesco Bernardini[1] and Vincenzo Manca[2]

[1] Dipartimento di Informatica
Università degli Studi di Pisa
Corso Italia 40, 56125 Pisa, Italy
bernardf@inwind.it
[2] Dipartimento di Informatica
Università degli Studi di Verona
Strada Le Grazie 15, 37134, Verona, Italy
vincenzo.manca@univr.it

**Abstract.** We propose a new variant of P Systems, based on *boundary rules*, shortly called PB Systems. *Boundary rules* means that rules are not internal to regions but they are able to "see" even externally, that is, they are sensible to what happens around the border of the membranes. We prove the computational universality for PB systems with three membranes which sense at most one symbol outside and one symbol inside a membrane. Finally, we investigate the relationships with the basic model of P systems by proving an equivalence between P systems without priority, thickness or dissolution operator, and PB systems that use communication rules of a restricted form.

## 1 Introduction

P Systems represent a class of distributed and parallel computing devices of a biological type introduced in [8] (see also [9]). Their basic ingredient is a *membrane structure*, consisting of several membranes embedded in a main membrane called *skin*. Each membrane delimits a *region*: the space between a membrane and all directly inner membranes, if any inner membrane exists. Each region contains a multiset of *objects*; the objects evolve according to given *evolution rules*, which are applied non-deterministically in a maximally parallel manner (at each step, in each region, all objects that can evolve must do it). The objects can also be communicated from a region to another one. Several variants of the basic model have been considered, which introduce in P systems further features, such as priority, membrane thickness, membrane dissolution and membrane division. Many of these variants have been proved to be computationally complete: they are able to generate all Turing computable sets of natural numbers. For a complete survey of Membrane Computing area we refer to [10].

Communication of objects through membranes is one of the most important ingredients of P systems. In the basic model, such a communication is controlled by the operators *here*, *in*, *out* that are associated with the objects produced by local rules. More precisely, a rule assumes typically the form $ca \rightarrow b_{in_j} d_{out} d_{here}$

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 107–118, 2003.

and it "says" that a copy of an object $c$ and a copy of an object $a$ are replaced by a copy of $b$ and two copies of $d$, where the copy of $b$ enters into membrane $j$, a copy of $d$ is sent out of the current membrane, and a copy of $d$ remains in the same region.

Recently, in [7] a new variant was proposed which introduces in membrane systems a form of communication based on a biochemical transport mechanism called *symport/antiport*. When two chemicals can pass through a membrane only together, in the same direction, the process is called *symport*; when the two chemicals pass only with the help of each other, but in opposite directions, we say that we have an *antiport* process (see [1]). More precisely, the symport mechanism is encoded by rules of the form $(ab, in)$, $(ab, out)$, whereas the antiport mechanism by rules of the form $(a, in; b, out)$.

Here, we generalize that idea of membrane transport by introducing explicitly membranes in the rules as "boundary entities" which control inside and outside of the delimited region. In these systems, we shortly call PB systems, communication is allowed by rules of the form $\alpha\,[_i\,\beta \rightarrow \alpha'\,[_i\,\beta'$. The meaning of such rules is the following: if the membrane $i$ contains the multiset $\beta$ and outside the membrane $i$ is present the multiset $\alpha$, then a "communication" can be established producing the multisets $\alpha'$,$\beta'$ outside and inside the membrane $i$, respectively. Moreover, with respect to [7], in PB systems, transformation rules, that will be written in the form $[_i\,\beta \rightarrow [_i\,\beta'$, are allowed. We remark that rules are no longer "located" into regions but, in a sense, they are located on the "boundaries" of membranes (the semi-bracket notation indicates a membrane, hence the localization of the rule).

## 2 Preliminaries

We recall some basic notions concerning formal language theory and the notation usually adopted in Membrane Computing area; for further details we refer to [11] and [10].

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 $\lambda$. 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 $M : V \longrightarrow N$ such that, $M(a)$ defines the *multiplicity of $a$ in the multiset $M$* ($N$ denotes the set of natural numbers); the symbols in $V$ are called *objects*. Such a multiset can be represented by a string $a_1^{M(a_1)}\,a_2^{M(a_2)}\ldots a_n^{M(a_n)} \in V^*$ and by all its permutations, $a_j \in V$, $M(a_j) \neq 0$, $1 \leq j \leq n$. In other words, we can say that each string $x \in V^*$ identifies a finite multiset over $V$ defined by $M_x = \{\,(a, |x|_a)\,|\,a \in V\,\}$.

Consider the language $C(V)$ over the alphabet $V \cup \{[, ]\}$, whose strings are recurrently defined as follows:

1. $[\,x\,] \in C(V)$, for each $x \in V^*$,
2. if $\mu_1, \ldots, \mu_n \in C(V)$, then $[\,x\,\mu_1 \ldots \mu_n\,] \in C(V)$, for each $x \in V^*$,
3. nothing else is in $C(V)$.

Now, let $\approx$ be the following relation over the elements of $C(V)$:

$$\mu \approx \mu' \text{ iff } \mu = \alpha\,\mu_1\,\mu_2\,\beta, \mu' = \alpha\,\mu_2\,\mu_1\,\beta \text{ or } \mu = \alpha\,[\,x\,\beta, \mu' = \alpha\,[\,y\,\beta,$$

with $\alpha, \beta \in (V \cup \{\,[\,,]\,\})^*$, $\mu_1, \mu_2 \in C(V)$, $x, y \in V^*$, $y$ a permutation of $x$. We also denote by $\approx$ the reflexive and transitive closure of $\approx$. This is clearly an equivalence relation. We denote by $\overline{C(V)}$ the set of equivalence classes of $C(V)$ with respect to the relation $\approx$. The elements of $\overline{C(V)}$ are called *configurations*.

Each matching pair of parentheses $[\,,]$ appearing in a configuration represents a *membrane*, whereas each string $x$ placed to the right of each open parenthesis defines the contents of the membrane that is identified by that parenthesis. The external membrane of a configuration is called *skin*. A membrane which appears in a configuration without any membrane inside is called an *elementary* membrane.

In the sequel, we will represent a configuration of a PB system by a string $\mu \in C(V)$ where both the order of neighboring membranes placed at the same level and the order of symbols inside the membranes does not matter.

## 3 P Systems with Boundary Rules

We are ready to define formally PB systems.

**Definition 1.** *A* PB system *is a construct*

$$\Pi = (V, \mu_0, R, i_O),$$

*where:*

1. $V$ *is an alphabet of symbols called* objects,
2. $\mu_0$ *is the* initial configuration *where* $m$ *is the number of membranes,*
3. $R$ *is a finite set of rules of the following forms:*
   $xx'\,[_i\,y'y \to xy'\,[_i\,x'y$, *for* $x, y, x', y' \in V^*$, $1 \le i \le m$;
   *if* $i = 1$, *then* $xx' = \lambda$
   *(*Communication rules*),*
   $[_i\,y \to [_i\,y'$, *for* $y, y' \in V^*$ *and* $1 \le i \le m$
   *(*Transformation rules*),*
4. $i_O \in \{1, \ldots, m\}$ *is the label of the* output membrane.

In this definition we can find the basic elements of every membrane systems: a structure with $m$ membranes containing $m$ multisets of objects that are associated with the regions delimited by the membranes (the initial configuration $\mu_0$), a finite set of rules $R$ and an output membrane $i_O$.

In the set $R$ we identify two types of rules: communication rules and transformation rules. A transformation rule $[_i\,y \to [_i\,y'$ allows the system to produce, in membrane $i$, a new multiset $y'$ starting from the multiset $y$, which is consumed by the rule. Instead, with communication rules of the form $xx'\,[_i\,y'y \to xy'\,[_i\,x'y$ we can move objects through membranes: if membrane $i$ contains the multiset

$y'y$ and outside membrane $i$ is present the multiset $xx'$, then the multiset $x'$ moves into membrane $i$ while the multiset $y'$ is sent out of the same membrane. Clearly, some of these multisets may be empty. In particular, in communication rules that refer to membrane 1, the multiset $xx'$ must be empty. It means that, in PB systems, we do not allow the skin membrane to interact with the environment and we suppose the objects sent out from membrane 1 to be lost.

As usual, these rules are applied in a nondeterministic and maximally parallel manner: the rules are chosen in a nondeterministic manner and this choice must be "exhaustive", that is, no other rule can be applied to the objects from the current configuration not assigned yet to rules (note that we do not count the use of membranes, any number of rules may involve the same membrane).

A *computation* in a PB system is a sequence of transitions among configurations of the form: $\mu_0 \Rightarrow \mu_1 \Rightarrow \mu_2 \Rightarrow \ldots$ where, for all $j > 0$, $\mu_j$ is the configuration obtained by applying rules of $R$ to the configuration $\mu_{j-1}$. A computation is successful if it *halts* with a configuration where no rules can be applied. The *result* of a successful computation is the multiset that is associated with the membrane $i_O$ in the halting configuration. A computation which never halts yields no result. Now, given a PB system $\Pi$, we say that $\Pi$ *generates the vector* $\Psi_V(x)$ (the *Parikh image of* $x$), for $x \in V^*$, iff, $x$ represents the result of a successful computation in $\Pi$; the set of all vectors generated by $\Pi$ is denoted by $Ps(\Pi)$.

At this point, we can identify new families of sets of vectors generated by PB systems. We denote by $PsPB_m(e, j, c, f)$, for $m > 0$, $e, j \geq 0$, $c \in \{S, nS\}$ and $f \in \{Coo, nCoo\}$ the family of all sets of vectors generated by PB systems with at most $m$ membranes such that:

- for all communication rules $xx' [_i y'y \rightarrow xy' [_i x'y$, we have: $0 \leq |xx'| \leq e$, $0 \leq |y'y| \leq j$, that is, in the left side of every communication rule we regard at most $e$ symbols outside a membrane and at most $j$ symbols inside it; we say also that communication rules are of type $(e, j)$;
- if $c = nS$, all communication rules assume the form:

$$[_i y'y \rightarrow y' [_i y \text{ or } xx' [_i \rightarrow x [_i x'$$

  that is, we have unidirectional communication that only depends either on the internal or the external contents of the membranes; instead, if $c = S$, communication rules of any type are allowed;
- if $f = nCoo$, transformation rules are non cooperative rules: on the left side of every rule there is only one symbol; instead, if $f = Coo$, cooperative transformation rules are allowed.

## 4 The Power of PB Systems

We prove that PB systems with bidirectional communication performed by rules of type $(1, 1)$ and only three membranes are able to characterize the family of recursively enumerable sets of vectors of natural numbers $PsRE$. In the proof

we need the notion of *matrix grammar with appearance checking*, for short "with a.c." (see [5]). A matrix grammar with a.c. is a construct $G = (N, T, S, M, F)$, where $N, T$ are disjoint alphabets, $S \in N$, $M$ is a finite set of sequences of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, with $A_i \to x_i$ a context free rule over $N \cup T$, for all $1 \leq i \leq n$, and $F$ a set of occurrences of rules in $M$. Given $w, z \in (N \cup T)^*$, we write $w \Longrightarrow z$ if and only if there is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n) \in M$ such that $w = w_1$, $z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w'_i A_i w''_i$, $w_{i+1} = w'_i x_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$, or $w_i = w_{i+1}$, $A_i$ does not appear in $w_i$, and the rule $A_i \to x_i$ appears in $F$ (the rules of a matrix are applied in sequence, possibly skipping the rules in $F$ if they cannot be applied; in this latter case we say that the rules in $F$ are applied in *appearance checking mode*).

The language generated by a matrix grammar with a.c. $G$ is defined by $L(G) = \{ w \in T^* \mid S \Longrightarrow^* w \}$. The family of languages of this form is denoted by $MAT_{ac}$; it is known that $MAT_{ac} = RE$ (see [5]).

A matrix grammar with a.c. $G = (N, T, S, M, F)$ is said to be in *binary normal form* if $N = N_1 \cup N_2 \cup \{S, \#\}$, with these three sets mutually disjoint, and the matrices in $M$ assume one of the following form:

1. $(S \to X\, A)$, for $X \in N_1$, $A \in N_2$,
2. $(X \to Y, A \to x)$, for $X, Y \in N_1$, $A \in N_2$, $x \in (N_2 \cup T)^*$,$|x| \leq 2$,
3. $(X \to Y, A \to \#)$, for $X, Y \in N_1$, $A \in N_2$,
4. $(X \to \lambda, A \to x)$, for $X \in N_1$, $A \in N_2$, $x \in T^*$.

There is only one matrix of type 1 and $F$ consists exactly of all rules $A \to \#$ appearing in matrices of type 3; # is a trap-symbol, once introduced, it is never removed. A matrix of type 4 is used only once, in the last step of the derivation. Note that, given a matrix grammar $G$, in every step of a derivation, the strings that are produced by applying the matrices in $M$ assume the form $\alpha w$, with $\alpha \in N_1 \cup \{\lambda\}$, $w \in (N_2 \cup T)^*$. That is, in every but the last step of a derivation, we have only one symbol in $N_1$ which is used to control the rewriting of symbols in $N_2$; the unique symbol in $N_1$ is deleted in the last step of the derivation by applying a matrix of type 4; no further matrix can be used after removing the symbol from $N_1$. It is known that for each matrix grammar there exists an equivalent matrix grammar in the binary normal form (see Lemma 1.3.7 in [5]).

We have the following result:

**Theorem 1.** $PsPB_3(1, 1, S, nCoo) = PsRE$.

*Proof.* Consider a matrix grammar with a.c. $G = (N, T, S, M, F)$ in binary normal form. Assume that matrices of type 2 and 4 are labeled, in a one-to-one manner, by $m_1, \ldots, m_k$, and that matrices of type 3 are labeled, in a one-to-one manner, by $m_{k+1}, \ldots, m_n$. We construct a PB system $\Pi$ that simulates the grammar $G$ such that $Ps(\Pi) = Ps(L(G))$.

Before defining formally such a PB system and describing its behavior, we briefly illustrate our idea of simulation. The PB system will have the following membrane structure:

$$[_1\, [_2\, [_3\, ]_3\, ]_2\, ]_1.$$

Membrane 1 contains all symbols in $N_2 \cup T$ that are produced by derivations in $G$, and membrane 2 contains the unique symbols in $N_1$ that is present in the strings produced by such derivations. Membrane 3 is an auxiliary membrane that is used to avoid interference among applications of different matrices. Now, consider a configuration of the form

$$[_1\, w\, [_2\, X [_3\, ]_3\, ]_2\, ]_1$$

with $w \in (N_2 \cup T^*)$, $X \in N_1$; then, suppose that we want to apply a matrix of type 2 or 4 of the form $m_i : (X \to \alpha, A \to \beta)$, with $\alpha \in N_1 \cup \{\lambda\}$, $\beta \in (N_2 \cup T)^*$, $|\beta| \leq 2$. In this case, in membrane 2, we assign the index $i$ to the symbol $X$ and we produce the auxiliary symbols $q_i$, $r_i$. The symbol $r_i$, by using a rule $A\, [_2\, r_i \to r_i\, [_2\, A$, takes the corresponding symbol $A$ and brings it from membrane 1 into membrane 2, while the symbol $q_i$ is moved into membrane 1 where it is replaced by $q_i'$. In the next step, in membrane 2 we assign an index $j$ to the symbol $A$ and we use $q_i'$ to check if $j = i$. In such a case, by using a rule $q_i'\, [_2\, A_i \to A_i\, [_2\, q_i'$, the symbol $A_i$ comes back into membrane 1. At this point, we complete the simulation of $m_i$ by replacing $A_i$ with $\beta$ and $X_i$ with $\alpha$. Instead, in the case of a matrix of type 3 of the form $m_i : (X \to Y, A \to \#)$, we assign the index $i$ to the symbol $X$ and we produce the auxiliary symbols $r_i'$, $t$. The symbol $r_i'$ is used to check if the symbol $A$ is contained in membrane 1 or not (the rule $A \to \#$ is applied in appearance checking mode). If membrane 1 contains the symbol $A$, an infinite computation that yields no result is generated, otherwise we just replace $X_i$ with $Y$ for completing the simulation of the matrix $m_i$. In the meanwhile the symbol $t$ is moved into membrane 3 and it is used to remove the symbol $r_i'$ from membrane 2.

This idea of simulating the matrices in $G$ is formalized by the following PB system:

$$\Pi = (V, \mu_0, R, i_O),$$

with:

$$\begin{aligned}
V &= N_1 \cup N_2 \cup T \cup \{X_i''', X_i'', X_i', X_i \mid X \in N_1,\ 1 \leq i \leq n\} \\
&\quad \cup \{A_i \mid A \in N_2,\ 1 \leq i \leq k\} \cup \{r_i, q_i', q_i \mid 1 \leq i \leq k\} \\
&\quad \cup \{r_i' \mid k+1 \leq i \leq n\} \cup \{Z, f, t', t\}, \\
\mu_0 &= [_1\, A\, [_2\, X\, [_3\, ]_3\, ]_2\, ]_1, \text{ for } (S \to XA) \text{ the unique matrix of type 1 in } M, \\
R &= \{[_1\, r_i \to r_i\, [_1\, ;\ [_1\, q_i \to [_1\, q_i'\, ;\ [_2\, q_i \to q_i\, [_2\, ;\ [_2\, r_i \to [_2\, Z \mid 1 \leq i \leq k\} \\
&\quad \cup \{[_1\, A_i \to [_1\, x\, ;\ [_2\, A \to [_2\, A_i\, ;\ [_2\, A_i \to [_2\, Z\, ;\ [_2\, X \to [_2\, X_i'''\, r_i\, q_i\, ; \\
&\qquad [_2\, X_i''' \to [_2 \to X_i''\, ;\ A\, [_2\, r_i \to r_i\, [_2\, A\, ;\ q_i'\, [_2\, A_i \to A_i\, [_2 q_i' \mid \text{ for} \\
&\qquad m_i : (X \to Y, A \to x) \in M \text{ or } m_i : (X \to \lambda, A \to x) \in M, 1 \leq i \leq k\} \\
&\quad \cup \{[_2\, X_i \to [_2\, Y \mid m_i : (X \to Y, A \to x) \in M \text{ or } m_i : (X \to Y, A \to \#) \in M, \\
&\qquad 1 \leq i \leq n\} \\
&\quad \cup \{[_2\, X_i \to [_2\, f \mid \text{ for } \ m_i : (X \to \lambda, A \to x) \in M, 1 \leq i \leq k\} \\
&\quad \cup \{[_1\, r_i' \to [_1\, Z\, ;\ r_i'\, [_3\, t' \to t'\, [_3\, r_i' \mid k+1 \leq i \leq n\}
\end{aligned}$$

$$\cup \{[_2 X \to [_2 X_i'' \, r_i' \, t \,;\, A \, [_2 r_i' \to r_i' \, [_2 A \mid \text{for } m_i : (X \to Y, A \to \#) \in M, \; k+1 \le i \le n\}$$
$$\cup \{[_2 X_i'' \to [_2 X_i' \,;\, [_2 X_i' \to [_2 X_i \mid X \in N_1, 1 \le i \le n\}$$
$$\cup \{A \, [_2 f \to f \, [_2 A \mid A \in N_2\} \cup \{[_j Z \to [_j Z \mid 1 \le j \le 2\}$$
$$\cup \{[_1 f \to [_1 Z \,;\, t \, [_3 \to [_3 t \,;\, [_3 t \to [_3 t'\},$$
$$i_O = 1.$$

The initial configuration of the system is represented by the string

$$[_1 \, A \, [_2 \, X \, [_3 \, ]_3 \, ]_2 \, ]_1,$$

with $X \in N_1$, $A \in N_2$, for $(S \to XA)$ the unique matrix of type 1 in $M$.

In general, suppose to have a configuration of the form

$$[_1 \, w \, [_2 \, X \, y \, [_3 \, z \, ]_3 \, ]_2 \, ]_1, \tag{1}$$

for $w \in (N_2 \cup T)^*$ (the terminal symbols and nonterminal symbols in $N_2$ produced by derivations in $G$), $X \in N_1$ (the unique symbols in $N_1$ that is present in strings produced by derivations in $G$), $y \in (\{ q_i' \mid 1 \le i \le k \} \cup \{t'\})^*$, $z \in \{ r_i' \mid k+1 \le i \le n \}^*$ (some auxiliary symbols produced during the simulation of the matrices in $G$). Given a configuration like (1), we can apply either a rule $[_2 X \to [_2 X_i''' \, r_i \, q_i$, $1 \le i \le k$, or a rule $[_2 X \to [_2 X_i'' \, r_i' \, t$, $k+1 \le i \le n$. Therefore, we distinguish two cases.

**Case 1:** we apply a rule $[_2 X \to [_2 X_i''' \, r_i \, q_i$, $1 \le i \le k$; it means that we want to simulate the application of a matrix of type 2 or 4, that is:

$$m_i : (X \to Y, A \to x) \text{ or } m_i : (X \to \lambda, A \to x).$$

In this case, by applying the rule $[_2 X \to [_2 X_i''' \, r_i \, q_i$ to configuration (1), we obtain the following configuration

$$[_1 \, w \, [_2 \, X_i''' \, r_i \, q_i \, y \, [_3 \, z \, ]_3 \, ]_2 \, ]_1. \tag{2}$$

If membrane 1 does not contain the corresponding symbol $A$, besides the rules $[_2 \, q_i \to q_i \, [_2$, $[_2 X_i''' \to [_2 X_i''$, we are obliged to apply the rule $[_2 \, r_i \to [_2 \, Z$ and we obtain the configuration

$$[_1 \, w \, q_i \, [_2 \, X_i'' \, Z \, y \, [_3 \, z \, ]_3 \, ]_2 \, ]_1.$$

At this point we have to continue to apply the rule $[_2 \, Z \to [_2 \, Z$, hence we get an infinite computation that yields no result. Instead, if membrane 1 contains the symbol $A$, we can apply both the rules $[_2 \, q_i \to q_i \, [_2$, $[_2 X_i''' \to [_2 X_i''$ and the rule $A \, [_2 \, r_i \to r_i \, [_2 \, A$. The system reaches the configuration

$$[_1 \, w_1 \, q_i \, r_i \, [_2 X_i'' \, A \, y \, [_3 \, z \, ]_3 \, ]_2 \, ]_1,$$

with $w = w_1 A$. In the next step, in membrane 1, we replace $q_i$ with $q'_i$ and we sent out of the system the symbol $r_i$; in membrane 2, we replace $X''_i$ with $X'_i$ and we assign an index $j$ to the symbol $A$ by applying a rule $[_2 A \to [_2 A_j$ with $1 \leq j \leq k$. Now, if $j \neq i$, we are obliged to apply the rule $[_2 A_j \to [_2 Z$ and we get again an infinite computation. Otherwise, if $j = i$, we can apply the rule $q'_i [_2 A_i \to A_i [_2 q'_i$ so that the symbol $A_i$ can arrive in membrane 1; in the meanwhile, in membrane 2, the symbol $X'_i$ is replaced by $X_i$. In this way, we obtain the configuration

$$[_1 w_1 A_i [_2 X_i q'_i y [_3 z ]_3 ]_2 ]_1. \tag{3}$$

At this point, we have the symbol $A_i$ in membrane 1, the symbol $X_i$ in membrane 2 and we can complete the simulation of the matrix $m_i$.

If $m_i : (X \to Y, A \to x) \in M$ is a matrix of type 2, we just apply the rules $[_2 X_i \to [_2 Y$, $[_1 A_i \to [_1 x$ and we obtain the configuration

$$[_1 w_1 x [_2 Y q'_i y [_3 z ]_3 ]_2 ]_1. \tag{4}$$

In this manner, we have correctly simulated the application of the matrix $m_i$. Moreover, by setting $w' = w_1 x$, $y' = q'_i y$ in (4), we obtain a configuration like (1) and we can restart to simulate the application of another matrix in $M$.

If $m_i : (X \to \lambda, A \to x) \in M$ is a matrix of type 4, then to configuration (4) we apply the rules $[_2 X_i \to [_2 f$,$[_1 A_i \to [_1 x$ and we obtain the configuration

$$[_1 w_1 x [_2 f q'_i y [_3 z ]_3 ]_2 ]_1.$$

Now, if membrane 1 contains only terminal symbols (the matrix $m_i$ has been applied in the last step of a derivation in $G$), the computation halts and we have correctly simulated a derivation in $G$. Otherwise, we apply, in sequence, the rules $A [_2 f \to f [_2 A$, $[_1 f \to [_1 Z$, for some $A \in N_2$, and an infinite computation is generated.

**Case 2:** we apply a rule $[_2 X \to [_2 X''_i r'_i t$, $1 + k \leq i \leq n$; it means that we want to simulate the application of a matrix of type 3, that is:

$$m_i : (X \to Y, A \to \#).$$

In this case, by applying the rule $[_2 X \to [_2 X''_i r'_i t$ to configuration (1), we obtain the following configuration

$$[_1 w [_2 X''_i r'_i t y [_3 z ]_3 ]_2 ]_1.$$

If membrane 1 contains the corresponding symbol $A$, then the rule $A [_2 r'_i \to r'_i [_2 A$ is applied and the symbol $r'_i$ arrives in membrane 1. In membrane 1, we apply the rule $[_1 r'_i \to [_1 Z$ and we get an infinite computation that yields no result. Otherwise, if membrane 1 does not contain the symbol $A$, the symbol $r'_i$ remains in membrane 2, the symbol $X''_i$ is replaced by $X'_i$ and the symbol $t$ is moved into membrane 3. In membrane 3 such a symbol is replaced by $t'$ while,

in membrane 2, the symbol $X_i'$ is replaced by $X_i$. In this way, the system reaches the configuration

$$[_1\, w\, [_2\, X_i\, r_i'\, y\, [_3\, t'\, z\, ]_3\, ]_2\, ]_1.$$

At this point we can complete the simulation of the application of the matrix $m_i$ by applying the rules $[_2\, X_i \rightarrow [_2\, Y$, $r_i'\, [_3\, t' \rightarrow t'\, [_3\, r_i'$. Once again, we have correctly simulated the behavior of the matrix $m_i$ and we can restart to simulate the application of another matrix in $G$.

Having in mind the previous discussion about the work of the system, we can say that the only way to get halting computations in $\Pi$ is to correctly simulate derivations in $G$, that is, $Ps(\Pi) = Ps(L(G))$. □

We continue to investigate the power of PB systems by comparing this new variant with the basic model of P systems in terms of generative capacity. We will prove that, when we use communication rules of a restricted form, PB systems are equal in power to P systems that do not use priority among rules and which do not use any operator for modifying the membrane structure. Moreover, communication rules of type $(1, 1)$ suffice in the simulation of P systems by means of PB systems. In other words, we can say that, passing from communication controlled by operators $in, here, out$ to boundary rules, the power of P systems is not increased; what really increases the power of P systems is bidirectional communication.

For a formal definition of the basic model of P systems, we refer to [10]. We denote by $PsP_m(f, nPri, t, n\delta)$, for $f \in \{Coo, nCoo, Cat\}$, $t \in \{tar, i/o\}$, the families of sets of vectors of natural numbers generated by the class of P systems that do not use a priority relation among rules and that do not use any operator for modifying the membrane structure.

**Lemma 1.** *For all $m > 0$, $PsP_m(Coo, nPri, tar, n\delta) \subseteq PsPB_m(1, 1, nS, Coo)$.*

*Proof.* Consider a P system $\Pi = (V, \mu, w_1, w_2, ..., w_m, R_1, R_2, ..., R_m, i_O)$ such that $Ps(\Pi) \in PsP_m(Coo, nPri, tar, n\delta)$, where the symbols assume the usual meaning. We construct a corresponding PB systems $\Pi' = (V', \mu_0, R, i_O)$ such that $Ps(\Pi') \in PsPB_m(1, 1, nS, Coo)$, where:

- $V' = \{a', a'' \mid a \in V\} \cup \{a_{in_j}, a_{out_j} \mid a \in V, 1 \leq j \leq m\} \cup V$,
- $\mu_0$ is the configuration obtained by replacing each symbol $[_i$ in $\mu$ with $[_i\, w_i$, for each $1 \leq i \leq m$,
- $R$ is a finite set of rules such that:
  - $[_i\, a' \rightarrow [_i\, a$; $[_i\, a'' \rightarrow [_i\, a' \in R$, for each $a \in V$,
  - $[_i\, a_{out_i} \rightarrow a_{out_i}\, [_i$; $a_{in_i}\, [_i \rightarrow [_i\, a_{in_i} \in R$, for each $a \in V$, $1 \leq i \leq m$,
  - $[_i\, a_{out_j} \rightarrow [_i\, a$; $[_i\, a_{in_i} \rightarrow [_i\, a \in R$, for each $a \in V$, $1 \leq i \neq j \leq m$,
  - for each rule $u \rightarrow v \in R_i$, $1 \leq i \leq m$, $u \in V^*$, $v \in (V \times \{in_j, here, out\})^*$, there exists a corresponding rule $[_i\, u \rightarrow [_i\, v' \in R$, where $v'$ is the string which contains: a symbol $a''$ if the pair $(a, here)$ appears in $v$, a symbol $a_{out_i}$ if $(a, out)$ appears in $v$, a symbol $a_{in_j}$ if $(a, in_j)$ appears in $v$ and $j$ is a directly inner membrane. If a rule $u \rightarrow v \in R_i$ contains a pair $(a, in_j)$ and $j$ is not a directly inner membrane, there exists no corresponding rule in $R$ because the rule $u \rightarrow v$ never applies.

Let $w_i$ be the multiset associated with membrane $i$, $1 \leq i \leq m$, in a given configuration. The application of a rule $u \to v$ in $R_i$ to $w_i$ is simulated by the following application of rules in $R$. We start by applying the corresponding rule $[_i\, u \to [_i\, v'$ which produces some objects $a''$, some objects $a_{out_i}$ and some objects $a_{in_{j_1}}, \ldots, a_{in_{j_h}}$, for some directly inner membranes $j_1, \ldots, j_h$. In the next step, in membrane $i$, we replace each object $a''$ with $a'$, while each object $a_{in_{j_t}}$, $1 \leq t \leq h$, is moved into membrane $j_t$ by using a rule of the form $a_{in_{j_t}}\, [_{j_t} \to [_{j_t}\, a_{in_{j_t}}$, and each object $a_{out_i}$ is sent out of membrane $i$ by using a rule of the form $[_i\, a_{out_i} \to a_{out_i}\, [_i$. Finally, we apply a rule $[_i\, a' \to [_i\, a$ for each object $a'$ in membrane $i$, a rule $[_{j_t}\, a_{in_{j_t}} \to [_{j_t}\, a$ for each object $a_{in_{j_1}}$ in each membrane $j_t$ and a rule $[_k\, a_{out_i} \to [_k\, a$ for each object $a_{out_i}$ in the directly upper membrane $k$.

We can repeat the previous discussion for each rule in $R_i$ and for each membrane $i$, $1 \leq i \leq m$. Thus, the only way to get halting computations in $\Pi'$ is to correctly simulate halting computations in $\Pi$, that is, $Ps(\Pi') = Ps(\Pi)$. □

**Lemma 2.** *For all $m > 0$, $PsPB_m(*, *, nS, Coo)) \subseteq PsP_m(Coo, nPri, tar, n\delta)$.*

*Proof.* Consider a P system with boundary rules $\Pi = (V, \mu_0, R, i_O)$ such that $Ps(\Pi) \in PsPB_m(*, *, nS, Coo)$. We construct a P system

$$\Pi' = (V, \mu, w_1, w_2, ..., w_m, R_1, R_2, ..., R_m, i_O)$$

such that $Ps(\Pi') \in PsP_m(Coo, nPri, tar, n\delta)$ where:

- $\mu$ is obtained from $\mu_0$ by removing each string from $V^*$ from the right side of each symbol $[_i$, $1 \leq i \leq m$,
- $w_i$, $1 \leq i \leq m$, is the multiset associated with the membrane $i$ in the initial configuration $\mu_0$,
- for each rule of the form $[_i\, y \to [_i\, y'$ in $R$, $1 \leq i \leq m$, there exists a rule $y \to v$ in $R_i$ with $v$ a string which contains a pair $(a, here)$ for each symbol $a$ in $y'$,
- for each rule $[_i\, y'y \to y'\, [_i\, y$ in $R$, $1 \leq i \leq m$, there exists a rule $y'y \to v$ in $R_i$ with $v$ a string which contains a pair $(a, here)$ for each symbol $a$ in $y$ and a pair $(a, out)$ for each $a$ in $y'$,
- for each rule $yy'\, [_i \to y\, [_i\, y'$ in $R$, $1 \leq i \leq m$, there exists a rule $yy' \to v$ in $R_k$ where $k$ is the membrane directly above membrane $i$, and $v$ is a string which contains a pair $(a, here)$ for each symbol $a$ in $y$ and a pair $(a, in_i)$ for each symbol $a$ in $y'$.

With this construction, it is easy to prove that $Ps(\Pi') = Ps(\Pi)$. □

As a consequence of Lemma 1 and Lemma 2, we can establish the following equivalence between P systems and PB systems:

**Corollary 1.** *For all $m > 0$ we have:*
$PsPB_m(*, *, nS, Coo) = PsPB_m(1, 1, nS, Coo) = PsP_m(Coo, nPri, tar, n\delta)$.

Finally, we extend the previous results to the other families $PsP_m(f, nPri, t, n\delta)$, with $f \in \{Coo, nCoo, Cat\}$, $t \in \{tar, i/o\}$.

**Lemma 3.** *For all $m > 0$, $f \in \{Coo, nCoo\}$ we have:*

1. $PsP_m(nCoo, nPri, tar, n\delta) = PsPB_m(1, 1, nS, nCoo)$;
2. $PsP_m(Cat, nPri, tar, n\delta) \subseteq PsPB_m(1, 1, nS, Coo)$;
3. $PsP_m(Cat, nPri, i/o, n\delta) \subseteq PsPB_m(1, 1, nS, Coo)$;
4. $PsP_m(f, nPri, i/o, n\delta) \subseteq PsPB_m(1, 1, nS, f)$.

In what concerns points (i), we can prove separately the two inclusions. The inclusion $PsP_m(nCoo, nPri, tar, n\delta) \subseteq PsPB_m(1, 1, nS, nCoo)$ can be obtained by applying the construction of Lemma 1 to P systems with non cooperative rules. The opposite inclusion can be obtained by applying the construction of Lemma 2 to PB systems with communication rules of type $(1, 1)$. Note that the more general inclusion $PsPB_m(*, *, nS, nCoo) \subseteq PsP_m(nCoo, nPri, tar, n\delta)$ cannot be established. In fact, when we apply the construction of Lemma 2 to a PB systems with communication rules of type $(e, j) \neq (1, 1)$, we always produce a P system with cooperative rules. Finally, point (ii), (iii) and (iv) follow directly from the inclusions $PsP_m(Cat, nPri, tar, n\delta) \subseteq PsP_m(Coo, nPri, tar, n\delta)$ and $PsP_m(f, nPri, i/o, n\delta) \subseteq PsP_m(f, nPri, tar, n\delta)$.

## 5 Conclusions

With the formalism based on boundary rules, we point out the fundamental role of membranes as *separators* and *filters*. In this way, we obtain a model of membrane systems closer to biological reality which has the advantage to simplify many notational aspects and to eliminate a certain level of asymmetry implicit in P systems where membranes are passive entities. In other words, we can say that PB systems strengthen the role of membranes providing a mechanism to define complex interaction pattern among membranes. PB systems could provide an adequate basis for a new kind of investigation essentially devoted to their dynamical aspects in a sense that is close to some basic property of biological systems. Life is a very complex phenomenon that is due to the cooperation of many interacting processes. Actually, the developments in *Natural* and *Molecular Computing* provide several discrete symbolic models which could be useful to model biological phenomena in terms of discrete dynamical systems (for a survey of discrete dynamical systems of biological relevance see [4]).

A preliminary step on investigating dynamical aspects of P systems was developed in [2,3], where we essentially deal with the aspect of *periodicity* (life is always related to temporal cycles where many parameters change periodically and some basic rhythms are preserved). More precisely, our analysis is focused on the individuation of "minimal" PB systems which exhibit interesting phenomena of periodicity in a given environment. Formally, the environment for PB systems is a sequence of multisets characterized by the property of *almost periodicity* [6]. Such a property seems to be very useful to model environments which in time provide some substances indispensable to ensure the "life". PB systems with environment, shortly called PBE systems, need an active role of membranes in the communication with the external world. In fact, interesting phenomena of

periodicity can be modeled that are related to the basic periodicity of an external "environment cycle".

## References

1. Alberts, B.: Essential Cell Biology. An Introduction to the Molecular Biology of the Cell. Garland Publ. Inc. London (1998)
2. Bernardini, F.: Analisi dinamica di sistemi a membrane. "Tesi di Laurea in Informatica" Pisa (2002)
3. Bernardini, F., Manca, V.: Dynamical Aspects of P Systems. BioSystems (to appear)
4. Bonanno, C., Manca, V.: Discrete Dynamics in Biological Models. Romanian Journal of Information Science and Technology **5** (1–2) (2002) 45–67
5. Dassow, J., Păun Gh.: Regulated Rewriting in Formal Languages. Springer-Verlag Berlin (1989)
6. Marcus, S., Păun, Gh.: Infinite (Almost Periodic) Words, Formal Languages and Dynamical Systems. Bulletin EATCS **54** (1994) 224–231
7. Păun, A., Păun, Gh.: The Power of Communication: P Systems with Symport/Antiport. New Generation Computing **20** (3) (2002) 295–306
8. Păun, Gh.: Computing with Membranes. Journal of Computer and System Sciences **61** (1) (2000) 108–143
9. Păun, Gh.: From Cells to Computers: Computing with Membranes (P Systems). BioSystems **59** (2001) 139–151
10. Păun, Gh.: Membrane Computing. An Introduction. Springer-Verlag Berlin (2002)
11. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer-Verlag, vol. 1–3, Heidelberg (1997)

# Parallel Rewriting P Systems without Target Conflicts*

Daniela Besozzi[1], Giancarlo Mauri[2], and Claudio Zandron[2]

[1] Università degli Studi dell'Insubria
Dipartimento di Scienze Chimiche, Fisiche e Matematiche
Via Valleggio 11, 22100 Como, Italy
daniela.besozzi@uninsubria.it
[2] Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
{mauri,zandron}@disco.unimib.it

**Abstract.** We consider rewriting P systems with parallel application of evolution rules, where no conflicts on the communication of objects can arise. Different parallelism methods are used and only rules which have the same target indication can be simultaneously applied to a common string. The computational power is analyzed, with respect to Lindenmayer systems, and some relations among different parallel rewriting P systems are studied. Some open problems are also formulated.

## 1 Introduction

The P systems were introduced in [7] as a class of distributed parallel computing devices of a biochemical type. The basic model consists of a membrane structure composed by several cell-membranes, hierarchically embedded in a main membrane called the skin membrane. The membranes delimit regions and can contain objects, which evolve according to given evolution rules associated with the regions. In one step all regions are processed simultaneously 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 prescribed regions, which are always specified by a target indication associated with each rule.
A computation device is obtained: we start from an initial configuration and we let the system evolve. The objects expelled through the skin membrane (or collected inside a specified output membrane) are the result of the computation.

A survey and an up-to-date bibliography can be found at the web address http://psystems.disco.unimib.it.

In this paper we consider rewriting P systems ([7], [10], [3]) and our aim is to extend the application of evolution rules from sequential rewriting to the

* Work partially supported by contribution of EU commission under The Fifth Framework Programme, project "MolCoNet" IST-2001-32008.

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 119–133, 2003.

parallel one (see also [1], [4], [5], [6]). This fact is also biologically motivated, as a cellular substance could be processed by many chemical reactions (each on a different site) at the same time.

Observe that using parallel rewriting means that at each step of a computation a string has to be processed, if possible, by more than one rule at the same time, according to the prescribed parallel rewriting method. So in parallel rewriting P systems we have a three-level massive parallelism, involving membranes, string-objects, and rules as well. On the other hand, if the rules we apply to the same string have mixed target indications, then we have consistency problems for the communication of the resulting string, as there might be contradictory indications about the region where the string should be at the next step.

This problem has been previously faced and solved with different strategies ([4]), for example, by counting the number of target indications of types $here, in, out$ appearing after the parallel application of rules, and then communicating the string to the region corresponding to the maximal number of indications. Another possibility consists in choosing as target region the one which corresponds to the indication (if existing) that appears exactly once after the parallel application of rules.

A different approach for facing the problem has been considered in [1], where the definition of *deadlock state* has been introduced to the aim of describing situations where rules with mixed target indications are applied at the same time to a common string. When a situation of deadlock arises for a string, then such a string is not sent to outer or inner regions but it remains inside the current membrane, though it will not be processed anymore by any other rule. Hence the deadlock state for that string causes its further processing and communication to be stopped.

In this paper we deal with P systems which use different parallelism methods for rewriting the strings, but here we assume (actually, we ensure) that no target conflicts can arise. When some rules could be applied to a common string according to a chosen parallelism method, then these rules are simultaneously applied on condition that they all have the same target indication. If this is not the case, then only the proper subset of rules which have the same target will be actually used. We also use order relations (see, e.g., [7], [10]) defined over the set of rules, which describe the priority of application of some rules with respect to other rules.

We analyze the computational power of this kind of P systems, comparing them to Lindenmayer systems, and we study the relations among P systems which use different parallelism methods.

## 2 Parallel Rewriting Methods

We denote by $V^*$ the free monoid generated by the alphabet $V$, the empty string is denoted by $\lambda$ and $V^+ = V^* \setminus \{\lambda\}$ is the set of non-empty strings over $V$. We will refer to [2] and [8] for other formal language theory notions.

In this section we present some different types of parallel rewriting methods for context-free rules. We always assume the condition that two or more rules are not allowed to rewrite a symbol at the same time, as in interactionless Lindenmayer systems ([8], [9]).

Specifically, in this paper we will use the following methods of parallel rewriting:

(U) A rewriting step with **unique parallelism** consists in the substitution of all occurrences of exactly one symbol according to exactly one rule, which is nondeterministically chosen from all rules that can be applied to that symbol. That is, given a string $w = x_1 a x_2 a x_3 \dots x_n a x_{n+1}$ with $x_i \in (V \setminus \{a\})^*, 1 \leq i \leq n+1$, and a context-free rule $r : a \to \alpha$, in one parallel rewriting step we obtain the string $w' = x_1 \alpha x_2 \alpha x_3 \dots x_n \alpha x_{n+1}$.

(S) A rewriting step with **symbol parallelism** is a step where for each symbol that can be the subject of a rewriting rule, all of its occurrences are substituted according to the same rule. That is, given some distinct symbols $a_1, \dots, a_n \in V$ and a string $w = x_1 a'_1 x_2 a'_2 x_3 a'_3 \dots x_m a'_m x_{m+1}$, with $a'_i \in \{a_1, \dots, a_n\}, 1 \leq i \leq m, m \geq n$, and $x_j \in (V \setminus \{a_1, \dots, a_n\})^*, 1 \leq j \leq m+1$, and given only one context-free rule for each symbol $r_1 : a_1 \to \alpha_1, \dots, r_n : a_n \to \alpha_n$ (nondeterministically chosen between all rules which can be applied to each symbol), in one step we obtain the string $w' = x_1 \alpha'_1 x_2 \alpha'_2 x_3 \alpha'_3 \dots x_m \alpha'_m x_{m+1}$, where $\alpha'_i \in \{\alpha_1, \dots, \alpha_n\}, 1 \leq i \leq m$, and $\alpha'_i = \alpha_k$ in $w'$ if and only if $a'_i = a_k$ in $w$ for some $k = 1, \dots, n$.

(M) In a rewriting step with **maximal parallelism**, all occurrences of all symbols (which can be the subject of a rewriting rule) are simultaneously rewritten by rules which are nondeterministically chosen in the set of all applicable rewriting rules. That is, if the string $w = x_1 a_1 x_2 a_2 x_3 a_3 \dots x_n a_n x_{n+1}$, with $w \in V^+, a_1, \dots, a_n \in V$ (not necessarily distinct) and $x_i \in (V \setminus \{a_1, \dots, a_n\})^*, 1 \leq i \leq n+1$, is such that there are no rules defined over symbols in the strings $x_1, \dots, x_{n+1}$, and there are some rules $r_1 : a_1 \to \alpha_1, \dots, r_m : a_m \to \alpha_m$, not necessarily distinct, then we obtain in one maximal parallel rewriting step the string $w' = x_1 \alpha_1 x_2 \alpha_2 x_3 \alpha_3 \dots x_m \alpha_m x_{m+1}$.

(T) As in $(E)T0L$ systems, we can consider the set of rewriting rules divided into subsets of rules, that is **tables** of rules. In this case, if we have a string $w$ and some tables $t_1 : [r_1^1 : a_{1,1} \to \alpha_{1,1}, \dots, r_{k_1}^1 : a_{1,k_1} \to \alpha_{1,k_1}], \dots, t_l : [r_1^l : a_{l,1} \to \alpha_{l,1}, \dots, r_{k_l}^l : a_{l,k_l} \to \alpha_{l,k_l}]$, then in one step only the rules from a table (which is nondeterministically chosen) can be applied, and these rules must be applied in parallel to all occurrences of all symbols in $w$, but not necessarily following the order the rules appear in the table. Moreover, if some rules in the chosen table are defined over symbols not in $w$, or if the number of rules in the table exceeds the length of the string, then we skip those (not defined or exceeding) rules without forbidding the application of the entire table.

## 3 Parallel Rewriting P Systems without Target Conflicts

A *parallel rewriting P system* of degree $n+1$ is defined by the construct

$$\Pi = (V, T, \mu, M_0, \ldots, M_n, (R_0, \rho_0), \ldots, (R_n, \rho_n)),$$

where:

1. $V$ is the alphabet of the system;
2. $T \subseteq V$ is the terminal alphabet;
3. $\mu$ is a membrane structure with $n+1$ membranes, which are injectively labeled by numbers in the set $\{0, 1, \ldots, n\}$. The skin membrane has always label 0;
4. $M_0, \ldots, M_n$ are finite languages over $V$, representing the strings initially present in the regions $0, 1, \ldots, n$ of the system;
5. $R_0, \ldots, R_n$ are finite sets of *evolution rules* of the form $a \to \alpha(tar)$, with $a \in V, \alpha \in V^*, tar \in \{here, out, in\}$, associated with the regions of $\mu$;
6. $\rho_0, \ldots, \rho_n$ are partial order relations defined over $R_0, \ldots, R_n$ respectively.

The application of evolution rules is performed as follows: in one step all regions are processed simultaneously by using the rules in a nondeterministic and parallel manner, according to the order defined by the partial relations (if existing). This means that in each region the objects to evolve and the rules to be applied to them are nondeterministically chosen, but all objects which can evolve should evolve. Moreover, at each step of a computation a string has to be rewritten by means of as many rules at the same time as possible; these rules are applied according to the chosen parallelism method.

The strings resulting after the parallel application of the rules must be communicated to the prescribed region, which is always specified by the target indication associated with each rule. For every region $i = 0, \ldots, n$ of the membrane structure we divide the set $R_i$ of evolution rules into mutually disjoint subsets of rules which have the same target indications, that is $R_i = \mathcal{H}_i \cup \mathcal{O}_i \cup \mathcal{I}_i$, where $\mathcal{H}_i = \{r \in R_i \mid tar(r) = here\}, \mathcal{O}_i = \{r \in R_i \mid tar(r) = out\}$ and $\mathcal{I}_i = \{r \in R_i \mid tar(r) = in\}$. Observe that for every inner region the set $\mathcal{I}_i$ will always be empty, and that for any other region any subset of rules could be empty as well.

Consider now some rules $r_1, \ldots, r_m$, for some $m \geq 2$, all of which can be applied to a common string $w$ at the same time. We assume that these rules will be applied to $w$ **only if** no conflicts arise for the target indications, that is only if it holds that (1) $r_1, \ldots, r_m \in \mathcal{H}_i$, or (2) $r_1, \ldots, r_m \in \mathcal{I}_i$, or (3) $r_1, \ldots, r_m \in \mathcal{O}_i$ (we apply in parallel only the rules which match on the target membrane). According to the chosen set of rules, the resulting string $w'$ (obtained after the parallel application of $r_1, \ldots, r_m$) (1) remains inside the current region $i$, (2) is communicated to a (nondeterministically chosen) inner region, (3) is communicated to the outer region. In particular, if the string exits the system, it can never come back and it may contribute to the output of the system.

At a given time the membrane structure together with all multisets of objects associated with the regions, represent the *configuration* of the system at that time. The $(n+2)$-tuple $C_0 = (\mu, M_0, \ldots, M_n)$ constitutes the initial configuration of the system. For two configurations $C_t = (\mu, M_0^t, \ldots, M_n^t), C_{t+1} = (\mu, M_0^{t+1}, \ldots, M_n^{t+1})$ of the system we say that there is a *transition* from $C_t$ to $C_{t+1}$ if we can apply the rules present in the regions according to the above prescriptions.

A sequence of transitions forms a *computation*. We say that a computation is *halting* when there is no rule which can be further applied in the current configuration. A computation is said to be *non-halting* if there is at least one rule which can be applied forever.

In this paper we consider *extended* P systems. The *output* of an extended system is the set of strings over $T$ (if any) sent out of the system during a computation, being it halting or not. Anyway, a string which exits the system but contains symbols not in $T$ does not contribute to the generated language.

We denote by $E\pi PRP_n$ the family of languages generated by extended rewriting P systems of degree at most $n$, where $\pi \in \{U, S, M, T\}$ denotes the used parallelism method. When also priority relations are used, then we use the notation $E\pi PRP_n(Pri)$. In the case when the number of membranes is not limited, then the subscript $n$ is replaced by $*$.

## 4 The Computational Power

In this section we analyze the computational power of parallel rewriting P systems, which use the four parallelism methods described in Section 2. Some lower bounds with respect to families of languages generated by Lindenmayer systems are given.

Further on we give some relations among P systems which use different parallel rewriting methods.

### 4.1 Relations with Lindenmayer Systems

A 0L system is a construct $G = (V, w, P)$, where $V$ is an alphabet, $w \in V^+$ is the axiom, and $P$ is a finite set of rules of the form $a \to v$, with $a \in V, v \in V^*$, such that for each $a \in V$ there is at least one rule $a \to v$ in $P$ (we say that $P$ is *complete*). For $w_1, w_2 \in V^*$ we write $w_1 \Longrightarrow w_2$ if $w_1 = a_1 \ldots a_n, w_2 = v_1 \ldots v_n$, for $a_i \to v_i \in P, 1 \leq i \leq n$. The language generated by $G$ is $L(G) = \{x \in V^* \mid w \Longrightarrow^* x\}$. If for each $a \in V$ there is exactly one rule $a \to v$ in $P$, then $G$ is said to be *deterministic*. If we distinguish a subset $T$ of $V$ and we define the generated language as $L(G) = \{x \in T^* \mid w \Longrightarrow^* x\}$, then $G$ is said to be *extended*. The family of languages generated by 0L systems is denoted by $0L$ ($E0L$ if the systems are extended, $ED0L$ if the systems are both extended and deterministic).

A *tabled* 0L system (T0L) is a construct $G = (V, w, P_1, \ldots, P_n)$ such that each triple $(V, w, P_i), 1 \leq i \leq n$, is a 0L system; each $P_i$ is called a *table*. The

language generated by $G$ is $L(G) = \{x \in V^* \mid w \Longrightarrow_{P_{j_1}} w_1 \Longrightarrow_{P_{j_2}} \dots \Longrightarrow_{P_{j_m}} w_m = x, m \geq 0, 1 \leq j_i \leq n, 1 \leq i \leq m\}$. A T0L system is deterministic when each table is deterministic, it is extended if there exists an alphabet of terminal symbols. The family of languages generated by T0L systems is denoted by $T0L$ ($ET0L$ if the systems are extended, $EDT0L$ if the systems are both extended and deterministic).

It is known that $CF \subset E0L \subset ET0L \subset CS$, where $CF, CS$ are the families of context-free and context-sensitive languages, respectively, and that $ED0L \subset EDT0L \subset ET0L, ED0L \subset E0L$. See [9] and [8] for other notions about Lindenmayer systems.

From a result in [1] we can state that:

**Theorem 1.** $ET0L \subseteq ETPRP_1$.

In [1] it has been (not directly) shown that, when considering P systems without deadlock, the family $ET0L$ is included in the family of languages generated by parallel rewriting P systems which use the maximal parallelism method and do not have an *a priori* bounded number of membranes. Here we improve that result by showing that two membranes suffice.

**Theorem 2.** $ET0L \subseteq EMPRP_2$.

*Proof.* According to Theorem 1.3 in [9], for each language $L \in ET0L$ there exists an ET0L system $G$ which generates $L$ and contains only two tables, that is $G = (V, T, w, P_1, P_2)$. At the first step of a derivation, we use table $P_1$. After using table $P_1$, we either use again table $P_1$ or we use table $P_2$, and after each use of table $P_2$ we either use table $P_1$ or we stop the derivation.

Making use of this observation, we construct the P system $\Pi = (V, T, [_0[_1]_1]_0, \emptyset, M_1, R_0, R_1)$ such that $L(\Pi) \in EMPRP_2$, where the initial multiset is $M_1 = \{w\}$ and the sets of rules are:

$$R_0 = \{A \to x(in) \mid A \to x \in P_2\} \cup \{A \to x(out) \mid A \to x \in P_2\};$$
$$R_1 = \{A \to x(here) \mid A \to x \in P_1\} \cup \{A \to x(out) \mid A \to x \in P_1\}.$$

The computation starts in membrane 1. Here we simulate the rules from table $P_1$ in $G$, by using in parallel all the rules with target *here* or all the rules with target *out*. In the first case, the process can be repeated, and we simulate the possibility of using table $P_1$ as many times as we want. In the second case, the resulting string is sent to the skin membrane, where we simulate table $P_2$. Again, we can apply in parallel only rules with target *in* or rules with target *out*. In the first case, the string returns to membrane 1 where we simulate the application of table $P_1$ once more. In the second case, the string exits the system and the computation halts. If the string consists of terminal symbols, then it contributes to the output, otherwise it is ignored. Hence we can correctly simulate each and every derivation in $G$ and generate each and every string generated by $G$. □

We begin the analysis of parallel rewriting P systems which use unique or symbol parallelism methods.

**Theorem 3.** $EDT0L \subseteq EUPRP_*(Pri)$.

*Proof.* Consider an EDT0L system $G = (V, T, w, P_1, \ldots, P_n)$ and construct the P system $\Pi = (V', T, \mu, M_0, \ldots, M_n, M_c, (R_0, \emptyset), (R_1, \rho_1), \ldots, (R_n, \rho_n), (R_c, \rho_c))$ such that $L(\Pi) \in EUPRP_*(Pri)$, where the alphabet is $V' = V \cup T \cup \{\bar{A} \mid A \in V\} \cup \{X\}$, with $T \subseteq V$ and $V, \{\bar{A} \mid A \in V\}, \{X\}$ mutually disjoint. The initial multiset is $M_0 = \{Xw\}$, while all other multisets are empty; the membrane structure consists of $n+1$ elementary membranes, all placed directly inside the skin membrane.

Observe that each table in $G$ is complete and deterministic hence, if $V = \{A_1, \ldots, A_k\}$, then each table is of the form $[A_1 \to x_1, \ldots, A_k \to x_k]$, for some $x_j \in V^*, 1 \leq j \leq k$. We use one membrane to simulate each table $P_1, \ldots, P_n$ in $G$; each membrane is labeled with a number $i \in \{1, \ldots, n\}$, in correspondence with the simulated table. Any membrane $m_i$ contains the following set of rules which undergo the given priority relation:

$$\begin{aligned} R_i = &\ \{A_j \to \bar{x}_j(here) \mid (A_j \to x_j) \in P_i, j = 1, \ldots, k\} \cup \{X \to X(out)\}; \\ \rho_i : &\ \{A_j \to \bar{x}_j(here) \mid (A_j \to x_j) \in P_i, j = 1, \ldots, k\} > (X \to X(out)). \end{aligned}$$

The strings $\bar{x}_j$ correspond to the strings $x_j$ in $P_i$, where all symbols $A \in V$ are substituted with the respective $\bar{A} \in V'$.

The skin membrane has rules given by:

$$R_0 = \{X \to X(in), X \to \lambda(out)\}.$$

Moreover, we need one more membrane $m_c$ to substitute all overlined symbols with the corresponding non overlined symbols in $V$:

$$\begin{aligned} R_c = &\ \{\bar{A} \to A(here) \mid \bar{A} \in V'\} \cup \{X \to X(out)\}; \\ \rho_c : &\ \{\bar{A} \to A(here) \mid \bar{A} \in V'\} > (X \to X(out)). \end{aligned}$$

The system works in the following way. The computation starts in the skin membrane, where we can nondeterministically choose a rule among $X \to X(in)$, $X \to \lambda(out)$. If the first rule is applied, then the string enters a nondeterministically chosen inner membrane. If it enters any membrane $m_i$, for $i = 1, \ldots, n$, then the simulation of the corresponding table $P_i$ in $G$ begins. According to the partial relation defined over $R_i$, during (at most) $k$ transitions we have to apply in parallel every rule $A_j \to \bar{x}_j(here), 1 \leq j \leq k$, over all occurrences of every symbol $A_k$ which appears in $w$. (It is necessary to use overlined symbols, in order to avoid that some symbol in $x_j$ could be rewritten at the next transition, hence yielding a wrong simulation of the table.) Only now we can apply the remaining rule $X \to X(out)$ in $m_i$, which sends the resulting string $X\bar{w}'$ back to the skin membrane.

Once more, if we apply the rule $X \to X(in)$ then the string $X\bar{w}'$ enters an inner membrane. If it enters any membrane $m_i, 1 \leq i \leq n$, then the only applicable rule is $X \to X(out)$, the string returns unchanged to the skin membrane. Otherwise, if it enters membrane $m_c$, then in (at most) $k$ transitions all overlined symbols are substituted, in parallel for all occurrences of every symbol,

with the corresponding non overlined symbols. Thanks to the partial relation in $m_c$, only when the substitution is complete the string $Xw'$ can return to the skin membrane by means of the rule $X \to X(out)$.

The process can be iterated. In any moment, inside the skin membrane we can apply the rule $X \to \lambda(out)$: the support symbol $X$ is erased and the current string exits the system. If it is a terminal string, then it contributes to the output, otherwise it is ignored. It follows that $L(G) = L(\Pi)$. □

Directly from Theorem 3 it follows that:

**Corollary 1.** $ED0L \subseteq EUPRP_3(Pri)$.

*Proof.* Observe that an ED0L system can be seen as an EDT0L system with only one table. It follows that the language generated by an ED0L system $G$ can be also generated by a system of type $EUPRP_3(Pri)$, as defined in Theorem 3, where there is only one membrane for the single table in $G$. □

The previous results about P systems with unique parallelism hold also for systems which use symbol parallelism method but no priority relations.

**Theorem 4.** $EDT0L \subseteq ESPRP_*$.

*Proof.* Consider an EDT0L system $G = (V, T, w, P_1, \ldots, P_m)$ and let $\Pi$ be P system $(V', T, \mu, M_0, \ldots, M_m, R_0, \ldots, R_m)$ such that $L(\Pi) \in ESPRP_*$, where the alphabet is $V' = V \cup T \cup \{X\}$, with $T \subseteq V$ and $X \notin V$, and the initial multiset is $M_0 = \{Xw\}$ (all other multisets are empty). The membrane structure consists of $m$ elementary membranes, labeled with numbers $i \in \{1, \ldots, m\}$, all placed inside the skin membrane. Each inner membrane $m_i$ is used for the simulation of the table $P_i$ in $G$.

The system contains the following sets of rules:

$$R_0 = \{X \to X(in), X \to \lambda(out)\};$$
$$R_i = \{X \to X(out)\} \cup \{A \to x(out) \mid (A \to x) \in P_i\}, \text{ for all } i = 1, \ldots, m.$$

The system works in a way very similar to the system defined in Theorem 3, but here there is no need of using overlined symbols nor priority relations over rules. The computation starts in the skin membrane, where we can nondeterministically choose a rule among $X \to X(in), X \to \lambda(out)$. If the first rule is used, the string $Xw$ enters any membrane $m_i$, for $i = 1, \ldots, m$, and the simulation of the corresponding table $P_i$ in $G$ begins. In one parallel rewriting step, we apply all the rules $A \to x(out)$ over all occurrences of every symbol $A \in V$ which appears in $w$ (observe that, as $G$ is deterministic, in $R_i$ there will be only one rule for every symbol in $V$). In parallel, we also use the rule $X \to X(out)$. The resulting string $Xw'$ goes back to the skin membrane and the process can be iterated.

At any moment, inside the skin membrane we can apply the rule $X \to \lambda(out)$: the support symbol $X$ is erased and the current string exits the system. If it is a terminal string, then it contributes to the output, otherwise it is ignored. It follows that $L(G) = L(\Pi)$. □

**Corollary 2.** $ED0L \subseteq ESPRP_1$.

*Proof.* Given an ED0L system $G = (V, T, w, P)$ (an EDT0L system with only one table $P$), we can define an equivalent P system $\Pi = (V', T, [_0]_0, M_0, R_0)$ such that $L(\Pi) \in ESPRP_1$, where the alphabet is $V' = V \cup T \cup \{X\}$, with with $T \subseteq V$ and $X \notin V$, and the initial multiset is $M_0 = \{Xw\}$.

The system contains the following set of rules:

$$R_0 = \{X \to X(here), X \to \lambda(out)\} \cup \{A \to x(here) \mid (A \to x) \in P\}.$$

The system works in the following way: as long as we apply all the rules with target *here*, we simulate the derivations in $G$. At any moment, the remaining rule $X \to \lambda(out)$ can be used alone (as target conflicts are forbidden) and the current string exits the system. If it a terminal string, then it is accepted as output, otherwise it is ignored. Hence the equality $L(G) = L(\Pi)$ holds. □

Also the converse of Theorem 4 is true:

**Theorem 5.** $ESPRP_* \subseteq EDT0L$.

*Proof.* Consider the system $\Pi = (V, T, \mu, M_0, \ldots, M_{n-1}, R_0, \ldots, R_{n-1})$, such that $L(\Pi) \in ESPRP_n$. We show how to construct an EDT0L system $G = (V', T, X, P_1, \ldots, P_m)$, which generates the same language as $\Pi$. The alphabet of non terminal symbols is $V' = V \cup T \cup \{X_i \mid i = 0, \ldots, n-1\} \cup \{X, \bar{X}, \#\}$, where $T \subseteq V$ and $V, \{X_i \mid i = 0, \ldots, n-1\}, \{X, \bar{X}, \#\}$ are mutually disjoint sets.

In order to correctly define the tables which simulate the application of rules in $\Pi$, we recall that only those rules which have equal targets can be simultaneously applied to a common string, and that in each table exactly one rule appears for each symbol of the alphabet. Now let $m_i$ be a generic membrane in $\mu$, for some $i \in \{0, \ldots, n-1\}$. This membrane can contain some strings $w \in V^+$, some other membranes and a set of rules $R_i$, which can be divided into three disjoint subsets of rules $\mathcal{H}_i \cup \mathcal{O}_i \cup \mathcal{I}_i$ (corresponding to rules with target indications *here*, *out*, and *in*, respectively).

For each membrane $m_i$ in $\mu$, for all $i = 0, \ldots, n-1$, we define a set of tables of $G$ of the following four types:

1. **Starting tables**: for each string $w \in M_i$ we define a table $[X \to X_i w, \{X_j \to \# \mid j = 0, \ldots, n-1\}, \{A \to A \mid A \in (V \cup T)\}, \# \to \#, \bar{X} \to \#]$.
2. **Simulation tables**: let $lab_{i(out)}, lab_{i(in)}$ be the sets of labels of the membranes placed outside and inside (if any) the membrane $m_i$. We first define the tables that simulates the application of rules in $\mathcal{H}_i$: $[X \to \#, \bar{X} \to \#, \# \to \#, X_i \to X_i, \{X_j \to X_j \# \mid j = 0, \ldots, n-1, j \neq i\}, \{A \to x \mid (A \to x) \in \mathcal{H}_i\}, \{A \to A \mid \text{for all } A \in (V \cup T) \text{ such that there are no rules in } \mathcal{H}_i \text{ defined over } A\}]$.
   Similarly, to simulate rules in $\mathcal{O}_i$ (with $i \neq 0$) we define a table for each $j \in lab_{i(out)}$: $[X \to \#, \bar{X} \to \#, \# \to \#, X_i \to X_j, \{X_k \to X_k \# \mid k = 0, \ldots, n-1, k \neq i\}, \{A \to x \mid (A \to x) \in \mathcal{O}_i\}, \{A \to A \mid \text{for all } A \in (V \cup T) \text{ such that there are no rules in } \mathcal{O}_i \text{ defined over } A\}]$.

Finally, to simulate rules in $\mathcal{I}_i$ we define a table for each $j \in lab_{i(in)}$: $[X \to \#, \bar{X} \to \#, \# \to \#, X_i \to X_j, \{X_k \to X_k\# \mid k = 0, \ldots, n-1, k \neq i\}, \{A \to x \mid (A \to x) \in \mathcal{I}_i\}, \{A \to A \mid \text{for all } A \in (V \cup T) \text{ such that there are no rules in } \mathcal{I}_i \text{ defined over } A\}]$.

3. **Ending tables**: if $m_i$ is the skin membrane, then the rules in $\mathcal{O}_i$ must be simulated with the tables $[X \to \#, \bar{X} \to \#, \# \to \#, X_i \to \bar{X}, \{X_k \to X_k\# \mid k = 0, \ldots, n-1, k \neq i\}, \{A \to x \mid (A \to x) \in \mathcal{O}_i\}, \{A \to A \mid \text{for all } A \in (V \cup T) \text{ such that there are no rules in } \mathcal{O}_i \text{ defined over } A\}]$.
4. **Control table**: $[\{A \to \# \mid A \in V \setminus T\}, \{a \to a \mid a \in T\}, \{X_j \to \# \mid j = 0, \ldots, n-1\}, \bar{X} \to \lambda, X \to \#, \# \to \#]$.

Observe that there are as many simulation and ending tables as all possible combinations of rules (with equal targets) from any $R_i$ are. In this way we can simulate the nondeterministic application of rules in $\Pi$ without target conflicts.

Let us see how a computation in $\Pi$ can be simulated by a derivation in $G$. At the first step of a derivation in $G$, only one starting table should be chosen, otherwise the trap symbol # will be introduced and never removed. The chosen starting table determines both the string and the membrane of $\Pi$ that will be simulated; in fact the axiom $X$ is rewritten as $X_i w$ (no other rules can be applied), for some $i = 0, \ldots, n-1$ and some $w \in M_i$.

At the second step of a derivation, only a simulation table (or also an ending table if $m_i$ is the skin membrane) can be chosen. Assume we use a simulation table. In this case, if the chosen table corresponds to the rules of a different membrane $m_j$, then the trap symbol is introduced. The only way to correctly continue the derivation is to choose one of the tables which simulate rules in $m_i$. In such a case, according to the target of the rules present in the table, the symbol $X_i$ is transformed into $X_j$, for $j$ being the label of the target membrane. Each applicable rule in $m_i$ is used in the table, moreover if in $R_i$ there is not at least one rule for each symbol in $V$, then in the simulation table we add a "mute" rule of the form $A \to A$ for all of the symbols for which there are no rules in $R_i$ (in this way the table is complete and still deterministic). After the use of a simulation table, only another simulation (or ending) table can be used, according to the support symbol that appears in the current string.

Let us assume now that an ending table has been chosen. This means that we are to simulate the application of rules with target *out* in the skin membrane. Hence, at the next derivation step in $G$, we have to check that the produced string consists only of terminal symbols. In order to assure that the next table to be used is the control table, in the ending tables we define the rule $X \to \bar{X}$: the symbol $\bar{X}$ introduces the trap symbol # in all tables except the control one. Apart from this particular rule, the ending tables are analogous to all other simulation tables.

Once we have produced a string of the form $\bar{X}w'$, for some $w' \in (V \cup T \cup \#)^*$, by means of the control table we can check that only symbols from $T$ appear in $w'$. If this is not the case, then the rules $A \to \#$ will introduce the trap symbol (that could already be present in the string). In any case, the string could be used for other derivation steps but it will never be accepted as belonging to the

generated language. Otherwise, if the string is terminal, then no trap symbols will be introduced and at the same time $\bar{X}$ will be erased.

It follows that with the EDT0L system $G$ we can correctly simulate each and every computation in $\Pi$, so $L(G) = L(\Pi)$. □

Theorems 4 and 5 can be summarized as follows:

**Corollary 3.** *$ESPRP_* = EDT0L$.*

### 4.2 Relations among Parallel P Systems

As already pointed out before, it is quite natural to compare the power of parallel systems which make use of different kinds of parallel rewriting methods. We show here that the family of languages generated using systems with maximal parallelism coincides with the family of languages generated by systems which make use of table parallelisms. We give some partial answers for the other type of parallelisms considered in this paper.

**Theorem 6.** *(i) $EMPRP_n \subseteq ETPRP_n$;*
*(ii) $ETPRP_n \subseteq EMPRP_*$.*

*Proof.* $(i)$. Consider a P system $\Pi$ such that $L(\Pi) \in EMPRP_n$. In order to prove the inclusion, construct an equivalent P system $\Pi'$, such that $L(\Pi') \in ETPRP_n$, which has the same alphabets and membrane structure as $\Pi$. The set of rules of any membrane in $\Pi$ can be divided into three subsets $\mathcal{H}_i, \mathcal{O}_i, \mathcal{I}_i$. It suffices to put all the rules of each subset inside a single table in the corresponding membrane of $\Pi'$.

$(ii)$. Let $\Pi = (V, T, \mu, M_0, \ldots, M_{n-1}, R_0, \ldots, R_{n-1})$ be a system such that $L(\Pi) \in ETPRP_n$. We assume that $m_0$ is the skin membrane in $\mu$. We show how to construct a P system $\Pi' = (V', T, \mu', M'_0, \ldots, M'_{m-1}, R'_0, \ldots, R'_{m-1})$, such that $L(\Pi') \in EMPRP_m$, which generates the same language as $\Pi$. The alphabet of $\Pi'$ is $V' = V \cup \{X, X_t, X_{here}, X_{in}, X_{out}, \dagger\}$, where $V \cap \{X, X_t, X_{here}, X_{in}, X_{out}, \dagger\} = \emptyset$.

Consider a generic membrane $m_i$ of $\Pi$, for any $i = 0, \ldots, n-1$, which can contain a set $M_i$ of strings, a set $R_i$ of tables of rules and, possibly, a set $\{m_{i,1}, \ldots, m_{i,h}\}$ of other membranes. We show how to simulate this generic membrane in the system $\Pi'$, and we point out that the simulation of all other membranes follows the same recursive description below.

The membrane $m'_i$ in $\Pi'$, corresponding to $m_i$ in $\Pi$, is obtained by replacing every string $w$ in $M_i$ with a string $Xw$, where $X$ is a symbol not in $V$. The set of rules of the membrane $m'_i$ will be:

$R'_i = \{X \to X_t(in), X_{here} \to X_t(in), X_{in} \to X(in), X_{out} \to X(out)\}$. (In the skin membrane ($i = 0$), the last rule is replaced with $X_{out} \to \lambda(out)$).

Inside $m'_i$, we add some new membranes denoted by $m^t_{i,1}, \ldots, m^t_{i,s}$, one for each table $t_r \in R_i$, for $r = 1, \ldots, s$. Each new membrane $m^t_{i,r}$ will contain the following rules:

$$R^t_{i,r} = \{A \to yX_{tar}(out) \mid A \to y(tar) \in t_r\} \cup \{X_t \to \lambda(out), X \to \dagger(out)\}.$$

Finally, we add the rule $X_t \to \dagger(out)$ in each membrane $m'_{i,1}, \ldots, m'_{i,h}$, which are all placed inside membrane $m'_i$ and correspond to the membranes $m_{i,1}, \ldots, m_{i,h}$ originally placed in membrane $m_i$. (Observe that, if $i \neq 0$, then the rule $X_t \to \dagger(out)$ will also be placed inside $m'_i$ because of the recursive construction of the system.)

From the construction of $\mu'$, it follows that the number of membranes we need in $\Pi'$ to simulate each membrane in $\Pi$ depends on the number of tables in each $R_i$, hence $m$ cannot be a priori bounded. Instead, we can say that the depth is increased from the value $k$ to the new value $k+1$.

Let us now see how the system works. Consider a string $Xw$ in membrane $m'_i$, at the first step of a computation we always have to apply the rule $X \to X_t(in)$, which introduces the new symbol $X_t$ and sends the string inside an inner membrane. If $X_t w$ enters a membrane $m'_{i,j}, 1 \leq j \leq h$, then the symbol $\dagger$ is introduced and that string will never contribute to the output. Instead, if $X_t w$ enters a membrane $m^t_{i,r}, 1 \leq r \leq s$, then the computation can proceed. In this way, we can nondeterministically perform the simulation of a table from the corresponding membrane $m_i$ in $\Pi$.

Inside membrane $m^t_{i,r}, 1 \leq r \leq s$, we simulate the rules of the table $t_r$: we apply all the rules $A \to yX_{tar}(out)$ which can be applied and which correspond to rules $A \to y(tar)$ belonging to $t_r$, and in parallel we delete the symbol $X_t$. If some rules can be applied, then the rewritten string $w'$ returns to membrane $m'_i$ and the computation can proceed; if no rules can be applied, then no support symbols will be present in the string $w$ and the computation halts in membrane $m'_i$. Observe that more than one symbol $X_{tar}$ can be introduced at this step of the computation, in fact many rules of the form $A \to yX_{tar}(out)$ could be simultaneously applied to the current string.

If the introduced symbols are $X_{here}$, then in membrane $m'_i$ the rule $X_{here} \to X_t(in)$ will introduce the symbol $X_t$ again, and the string is ready for a new simulation of a table from $m_i$.

If the introduced symbols are $X_{in}$, then in membrane $m'_i$ the rule $X_{in} \to X(in)$ will introduce the symbol $X$ again. If the string enters a membrane $m'_{i,j}$, then the computation proceeds, otherwise it stops with the introduction of $\dagger$ in any other membrane.

If the introduced symbols are $X_{out}$, then in membrane $m'_i$ the rule $X_{out} \to X(out)$ will introduce the symbol $X$ again, and the string will exit the current membrane. In particular, if $m'_i$ is the skin membrane of $\Pi'$, then no support symbol will be introduced, the string will exit the system and, if it is a terminal string, it will contribute to the output, otherwise it will be ignored.

Hence we can correctly simulate every computation in $\Pi$ by a computation in $\Pi'$, it follows that $L(\Pi') = L(\Pi)$. □

An immediate consequence of the previous theorem is:

**Corollary 4.** $EMPRP_* = ETPRP_*$.

**Theorem 7.** $EUPRP_n(Pri) \subseteq ETPRP_*$.

*Proof.* Consider the system $\Pi = (V, T, \mu, M_0, \ldots, M_{n-1}, (R_0, \rho_0), (R_1, \rho_1), \ldots, (R_n, \rho_{n-1}))$, such that $L(\Pi) \in EUPRP_n(Pri)$. The system $\Pi'$ (such that $L(\Pi') \in ETPRP_*$), which generates the same language as $\Pi$, can be built as follows.

Consider a generic membrane $m$ in $\Pi$ containing $k$ rules (in what follows, $tar \in \{in, out, here\}$). The corresponding membrane $m'$ in $\Pi'$ is obtained by adding $k$ membranes immediately inside $m$, one for each rule to be simulated. Each string $w$ in $m$ is replaced with $Xw$ in $m'$, where $X \notin V$.

Then, we replace each rule $r_j : A \to x(tar)$ in $m$ with a corresponding table $T_j : [X \to \langle X_j, tar\rangle, \{B_i \to \# \mid \text{ for all rules } (B_i \to x_i) > (r_j : A \to x)\}, \langle X, tar\rangle \to \#; in]$. Moreover, we add the tables $[\langle X, tar\rangle \to X; tar]$, for all $tar \in \{in, out, here\}$, and $[\langle X_h, tar\rangle \to \#; here]$, for all $h \in \{1, \ldots, k\}$.

The added $j$-th internal membrane, $1 \leq j \leq k$, contains the table $[\langle X_j, tar\rangle \to \langle X, tar\rangle, \{X_i \to \# \mid i = 1, \ldots, k, i \neq j\}, r_j : A \to x; out]$.

The computation proceeds as follows: consider a generic membrane $m'$. We nondeterministically choose one table to be applied to the string; table $T_j$ simulates the rule $r_j$.

If the string contains some symbols which could be the subject of a rule in $\Pi$ with a higher priority, then the trap symbol is introduced. Otherwise, we introduce the symbol $\langle X_j, tar\rangle$ and we send the string to an inner membrane.

If the string reaches the added membrane which correspond to the simulation of the rule $r_j$, then we simulate the production and we send back to membrane $m$ a string of the form $\langle X, tar\rangle w'$. Here, we replace the symbol $\langle X, tar\rangle$ with $X$ and we send the obtained string to (one of) the target membrane corresponding to $tar$ (if we are in the skin membrane and the target is $out$, then $X$ is erased, that is we use the table $[\langle X, out\rangle \to \lambda; out]$ instead of $[\langle X, out\rangle \to X; out]$).

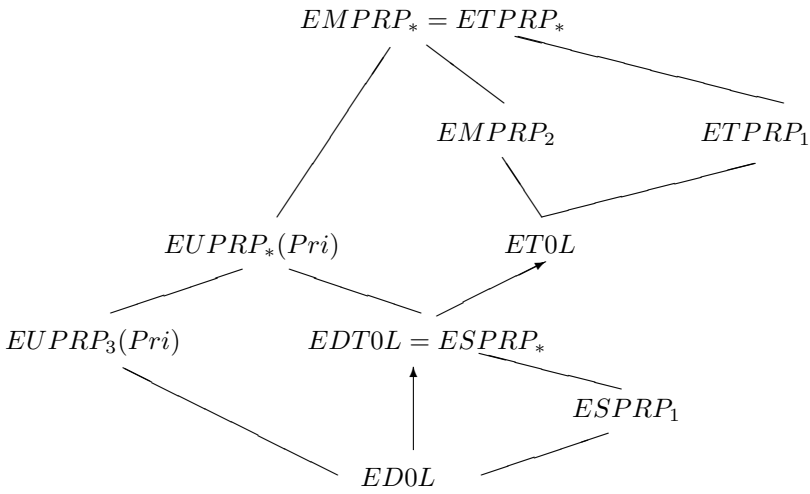
If the string is sent to "wrong" internal membranes (that is, to added membranes which simulate different rules or membranes originally present in $\mu$) then the trap symbol is introduced. The same holds if a string of the form $\langle X, tar\rangle w$ is the subject of a table $T_j : [X \to \langle X_j, tar\rangle, B_i \to \#, \langle X, tar\rangle \to \#; in]$.

It is easy to see that the language generated by $\Pi'$ is the same as that generated by $\Pi$. □

**Theorem 8.** $ESPRP_* \subset ETPRP_1$.

*Proof.* As $EDT0L \subset ET0L$, this inclusion directly follows from Theorem 1 and Corollary 3. □

We conclude this section by summarizing the previous results in the following hierarchy diagram: if two families are connected by a line, then the lower family is included in the upper family; if two families are connected by an arrow, then the lower family is properly included in the upper family. If two families are not connected, then they are not necessarily incomparable.




## 5 Final Remarks

We have considered P systems with string objects in which rules are applied in parallel, not only in all membranes and to all strings as it happens in usual P systems, but on symbols in the string, too.

We have introduced different kinds of parallelism and we have compared the generative power of systems using different parallelism methods with respect to various Lindenmayer systems (ET0L, ED0L, and EDT0L). As already pointed out in [1], some kind of parallelism are powerful enough to generate all ET0L languages using a small number of membranes. We improved one such result here, showing that two membranes suffice when we use systems with a maximal parallel application of the rules. Other kinds of parallelism are shown to be less powerful.

Moreover, we have compared the power of different parallelism methods. For instance, we have shown that the family of languages generated by P systems with a maximal parallel application of the rules coincides with the family of languages generated by P systems with table parallelism, while the family of languages generated by P systems with parallel application of rules on a single symbol is strictly included in the family of languages generated by P systems with table parallelism and with exactly one membrane.

Many problems remain open, mainly concerning comparisons among systems with different parallelism and their relations with respect to Lindenmayer systems.

## References

1. D. Besozzi, C. Ferretti, G. Mauri, C. Zandron, Parallel Rewriting P Systems with Deadlock, Pre-Proceedings of DNA8 Conference (M. Hagiya, A. Ohuchi, eds.), Hokkaido University, Japan, June 2002, 171–183.

2. J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
3. C. Ferretti, G. Mauri, C. Zandron, G. Păun, On Three Variants of Rewriting P Systems, *Theoretical Computer Science*, to appear.
4. S.N. Krishna, *Languages of P systems: Computability and Complexity*, PhD Thesis, 2001.
5. S.N. Krishna, R. Rama, A Note on Parallel Rewriting in P Systems, *Bulletin of the EATCS*, 73 (February 2001), 147–151.
6. S.N. Krishna, R. Rama, On the Power of P Systems Based on Sequential/Parallel Rewriting, *Intern. J. Computer Math.*, 77, 1–2 (2000), 1–14.
7. Gh. Păun, Computing with Membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (see also *Turku Center for Computer Science-TUCS Report* No 208, 1998, www.tucs.fi).
8. G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, Springer-Verlag, Heidelberg, 1997.
9. G. Rozenberg, A. Salomaa, *The Mathematical Theory of L Systems*, Academic Press, New York, 1980.
10. C. Zandron, *A Model for Molecular Computing: Membrane Systems*, PhD Thesis, 2001.

# Evolution–Communication P Systems

Matteo Cavaliere*

Research Group in Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tarraco 1, 43005 Tarragona, Spain
mc1.doc@estudiants.urv.es

**Abstract.** We propose a new class of P systems that use simple evolution rules (classical evolution rules without communication targets) and symport/antiport rules (for communication).
This type of P system is realistic for (at least) three different reasons: we do not have target indications in the evolution rules, we use very simple symport/antiport rules to realize communication, and we do not need objects available in the environment at the beginning of a computation. Somewhat expected, this new variant is still universal. We prove the universality in two cases: when using catalytic rules (but only one catalyst), symport/antiport rules of weight one, and two membranes, and when we use three membranes, symport/antiport rules of weight one, and no catalyst. Especially the latter result is of interest, because the catalysts were used in most universality proofs for P systems with symbol-objects. Also new is the proof technique we use: we start from programmed grammars with unconditional transfer.

## 1 Introduction

We introduce and investigate here a new class of P systems that joins two basic variants of P systems: the one with evolution rules, the other one with symport/antiport rules (in what follows, we assume the reader familiar with the basic elements of membrane computing, for instance, as presented in [5]). Moreover, in this variant, the evolution rules lose their target indications (in this way, the model becomes more realistic from a biological point of view) and the system is embedded in an empty enviroment (and not infinite as in the case of systems with symport/antiport rules). The idea of this new variant is to "split" the work of a P system in two phases: the *evolution* of the symbol-objects (application of evolution rules) and the *communication* between the regions of the system (application of the symport/antiport rules). In such a new model, we can have many different strategies to define a computation; in this paper we study the evolution-communication P systems where the computation takes place with a *mixed* "sequence" of evolution rules and symport/antiport rules. We show that

* This work was possible thanks to a research grant Beca URV from Rovira i Virgili University.

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 134–145, 2003.

such a variant, using non-cooperative rules, low passage of information (symport/antiport rules of weight equal to one) and two membranes can generate more than the Parikh image of context-free languages. Moreover, we show that, if we use one catalyst, low symport/antiport rules (symport/antiport rules of weight equal to one) and two membranes, then we can generate all the recursively enumerable sets of integer numbers. This result is interesting in view of the fact that in [6] it has been shown that a catalytic system, using six catalyst objects and two membranes, is universal. Here, we decrease the number of catalyst objects to one, at the price of using communication by mean of simple symport/antiport rules (of weight one). Also, we show that, if we use three membranes, then we can generate all the recursively enumerable sets of integer numbers using simple symport/antiport rules of weight one and simple non-cooperative evolution rules – hence no catalyst. The proof is based on simulating programmed grammars with unconditional transfer by means of P systems. As far as we know, it is the first time when the programmed grammars are used in membrane computing proofs. Up to now, the "standard" tools for obtaining universality results were matrix grammars with appearance checking and register machines.

We recall that another approach to avoid the use of target indications in the evolution rules, and having communication mixed with evolution, has been studied in [1]; However, while in [1] one changes the *structure* of the evolution rules, here we put together two already well-studied variants of P systems.

## 2 EC P Systems

We define the new variant of P systems, as asystems that use evolution rules as defined in [5], chapter 3 (but without communication targets, or, equivalently, with all the communication targets fixed as "here") and symport/antiport rules, as defined in [5], chapter 4. For simplicity, we often call the evolution rules without communication targets *simple evolution rules* (or *simple catalytic rules*).

**Definition 1.** *An* evolution-communication P system *(in short,* an EC P system*), of degree $m \geq 1$, is defined as*

$$\Pi = (O, \mu, w_1, w_2, \cdots, w_m, R_1, \cdots, R_m, R'_1, \cdots, R'_m, i_o),$$

*where:*

- *$O$ is the alphabet of objects;*
- *$\mu$ is a membrane structure with $m$ membranes (and hence $m$ regions) injectively labelled with $1, 2, \cdots, m$;*
- *$w_i$ are strings which represent multisets over $O$ associated with the regions $1, 2, \cdots, m$ of $\mu$;*
- *$R_i$, $1 \leq i \leq m$, are finite sets of simple evolution rules over $O$; $R_i$ is associated with the region $i$ of $\mu$; a simple evolution rule is of the form $u \to v$, where $u$ and $v$ are strings over the alphabet $O$;*

- $R'_i$, $1 \leq i \leq m$, *are finite sets of symport/antiport rules over* $O$; $R'_i$ *is associated with the membrane* $i$ *of* $\mu$;
- $i_o \in \{0, 1, 2, \cdots, m\}$ *is the output region; if* $i_o = 0$, *then it is the environment, otherwise* $i_o$ *is the label of an elementary membrane of* $\mu$.

The $m$-tuple of multisets of objects present at any moment in the regions of $\Pi$ represents the configuration of the system at that moment (the $m$-tuple $(w_1, \cdots, w_m)$ is the initial configuration). A transition between configurations is governed by the mixed application of the evolution rules and of the symport/antiport rules. All objects which can be the "subject" of the rules from the sets $R_i, R'_j$, $1 \leq i \leq m, 1 \leq j \leq m$, have to evolve by such rules. As usual, the rules from $R_i$ are applied to objects in region $i$ and the rules from $R'_i$ govern the communication of objects through membrane $i$. There is no difference between evolution rules and communication rules: they are chosen and applied in the non-deterministic maximally parallel manner. The system continues parallel steps until there remain no applicable rules (evolution rules or symport/antiport rules) in any region of $\Pi$. Then the system halts, and we consider the number of objects contained in the output region $i_o$, at the moment when the system halts, as the result of the computation of $\Pi$. This way to have a computation in an EC P system will be called the *mixed approach*.

We use the notation

$$NECP_m(i, j, \alpha), \alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 0\}$$
$$(PsECP_m(i, j, \alpha), \alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 0\})$$

to denote the family of sets of natural numbers (the family of sets of vectors of natural numbers) generated by EC P systems with at most $m$ membranes (as usually, $m = *$ if such a number is unbounded), using symport rules of weight $i$, antiport rules of weight $j$, and simple evolution rules that can be cooperative ($coo$), non-cooperative ($ncoo$), or catalytic ($cat_k$), using at most $k$ catalysts.

## 3 Computational Power: Preliminary Results

Before presenting the universality of EC P systems, we discuss some results which either follow directly from the definitions, or shed some light about the power of systems with a small number of membranes. We are interested in computational results about EC P systems that use symport/antiport rules of a low weight (systems with "long enough" symport/antiport rules are universal [5]) and with not "too powerful" evolution rules (we know that with cooperative rules or with catalyst rules, with "enough" catalyst objects, we have already universality [5], [6]). So, we will try to stay in the middle, using a combination of not so powerful symport/antiport rules and simple evolution rules. We recall that $NOP_m(\alpha, tar), PsOP_m(\alpha, tar), \alpha \in \{coo, ncoo\} \cup \{cat_k \mid k \geq 0\}$, denote the family of sets of natural numbers and the family of sets of vectors of natual numbers generated by symbol-object P systems of degree at most $m$, using rules of the type $\alpha$, and target communications of the type $\{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$.

As above, we use $cat_k$ to indicate that the corresponding P system uses at most $k$ catalysts.

Because communication targets can, obviously, simulate the control of communication as done by a symport rule of weight 1, we have:

$$NECP_m(1,0,\alpha) \subseteq NOP_m(\alpha, tar),$$

for all $\alpha \in \{coo, ncoo\} \cup \{cat_k \mid k \geq 0\}$.

From [6] we know that $NOP_2(cat_6, tar) = NRE$, and from [5] we have $NOP_m(ncoo, tar) = NCF$, for all $m \geq 1$. (As usual, if $FL$ is a family of languages, then we denote by $PsFL$ the family of Parikh images of languages in $FL$ and by $NFL$ the family of length sets of languages in $FL$.)

We know that, in the case of $NOP_1(ncoo, tar)$, the role of the target indications is only to send out (this means to "lose") some symbol-objects, using target indications of the form $a_{out}$, where $a$ is a symbol-object. We can simulate the evolution rules which contain such a target indication by replacing each $a_{out}$ with $\lambda$; in this way, we can avoid to use target indications. Thus, we have

$$NOP_1(ncoo, tar) \subseteq NECP_1(0,0,ncoo) \subseteq NECP_1(1,0,ncoo).$$

Therefore $NECP_1(1,0,ncoo) = NCF$.

Such an equality is no longer valid if we consider the Parikh sets and we use more powerful ingredients: symport/antiport rules of weight one and two membranes.

In the next theorem we show that, using two membranes, non-cooperative rules, and symport/antiport rules of weight one, we can generate more than the Parikh images of the context-free languages.

The idea of the proof is to simulate a programmed grammar without appearance checking (we know [2] that such grammars generate non semilinear languages, hence languages whose Parikh image is not in $PsCF$). The family of languages generated by programmed grammars with $\lambda$ rules and without appearance checking is denoted by $PR$.

**Theorem 1.** $PsPR \subseteq PsECP_2(1,1,ncoo)$.

*Proof.* Consider the programmed grammar $G = (N, T, P, S)$ without appearance checking. We denote by $l(S)$ the set of labels of rules of the form $(k : S \to x, \sigma(k)) \in P$. We add to $P$ the triple $(0 : U \to S, \sigma(0))$ with $\sigma(0) = l(S)$, where $U$ is a new non-terminal. We denote by $G'$ the obtained grammar $(N', T, P', U)$, where $N' = N \cup \{U\}$. Clearly, $L(G) = L(G')$ and each derivation in $G'$ starts with the rule with label 0.

For each $x \in N'$ we consider a new symbol $\bar{x}$; denote by $\bar{N}$ the set of such symbols, and define the morphism $\gamma : (N' \cup T)^* \longrightarrow (\bar{N} \cup T)^*$ in the following way: $\gamma(x) = \bar{x}$, if $x \in N'$, and $\gamma(x) = x$, if $x \in T$.

We construct the EC P system

$$\Pi = (O, \mu, w_1, w_2, R_1, R_2, R_1', R_2', i_o),$$

with:

$$
\begin{aligned}
O &= N' \cup T \cup \bar{N} \cup \{d_i, d'_i \mid i \in lab(P')\} \cup \{\#, f, c, c_1\},\\
\mu &= [_1[_2\,]_2]_1,\\
w_1 &= fU,\\
w_2 &= c,\\
i_o &= 0,\\
R_2 &= \{X \to \gamma(x)d_k \mid (k : X \to x, \sigma(k)) \in P'\} \cup \{d_i \to \# \mid i \in lab(P')\}\\
&\quad \cup \{\# \to \#, c_1 \to c\} \cup \{Y \to \bar{Y} \mid Y \in N'\},\\
R_1 &= \{d_i \to d'_i \mid i \in lab(P')\} \cup \{c \to c_1\} \cup \{\bar{Y} \to Y \mid Y \in N'\},\\
R'_2 &= \{(d_i, out; d'_j, in) \mid i \in \sigma(j)\} \cup \{(d_0, out; f, in), (c_1, in)\}\\
&\quad \cup \{(a, out) \mid a \in T\} \cup \{(c, out; Y, in), (\bar{Y}, out) \mid Y \in N'\},\\
R'_1 &= \{(a, out) \mid a \in T\}.
\end{aligned}
$$

The EC P system $\Pi$ simulates the programmed grammar $G'$ in the following way. The evolution rules simulate the context-free rules of $G'$, while the symport/antiport rules check if the evolution rules are applied according to the order defined by the labels in the success fields of $G'$. The symbol-object $c$, together with (one of) the antiport rules $(c, out; Z, in)$, for some $Z \in N'$, associated with membrane 2 are used; this makes sure that in each step only one object $Z \in N'$ is present in region 2, and, therefore, only one simple evolution rule of the form

$$Z \to \gamma(z)d_m, \text{ for } (m : Z \to z, \sigma(m)) \in P',$$

can be applied.

Assume that such a rule is applied in region 2 and the object $d_m$ is produced. This object indicates that the rule of $G'$ with label $m$ has been applied.

After the object $d_m$ has been produced, it must exit from region 2 (because of the evolution rule $d_m \to \#$ that produces the trap-symbol #). But the object $d_m$ can exit only if the "right" predecessor is present in region 1 ("right" in the sense of the order defined by the success fields of $G'$). In fact, to exit from region 2, the object $d_m$ must use (one of) the antiport rules associated with membrane 2, that is, $(d_m, out; d'_j, in)$, for $j$ such that $m \in \sigma(j)$. This antiport rule checks if the simple evolution rule has been applied in the correct order, as defined by the programmed grammar (the object $f$ is used only to start the computation).

When the object $d_m$ goes to region 1, it is changed in $d'_m$ and this object will be the new predecessor (this means that it "stores" the label of the last applied rule) and the computation will be continued. The evolution rules $Y \to \bar{Y}$, for $Y \in N'$, present in region 2, makes sure that when the computation halts, there are no objects of $N'$ present in regions 1 or 2.

In each step, each produced terminal is sent out by the symport rules associated with membrane 1 and 2, and we collect the output in the enviroment. Thus, modulo the order of symbols, the system $\Pi$ sends out exactly the strings of $L(G) = L(G')$. □

**Corollary 1.** $PsCF \subset PsECP_2(1,1,ncoo)$.

*Proof.* Because [2] $PsCF \subset PsPR$, the theorem implies the corollary. □

## 4 Universality of EC P Systems: A First Result

In this section we give a first universality theorem for EC P systems. This result is not a surprise, because the model joins two different models that are known to be universal. The interesting fact is that we can prove that our systems are universal even when they use very small symport/antiport rules, only one catalyst, and two membranes. The proof is based on the simulation of programmed grammars with appearance checking and with unconditional transfer. Such a grammar is an usual programmed grammar $G = (N, T, P, S)$, with the rules of $P$ of the form $(i : A \to x, \sigma(i), \varphi(i))$ with $\sigma(i) = \varphi(i)$ (the success and the failure fields coincide). We recall from [3] that such grammars are able to generate the recursively enumerable one-letter languages.

**Theorem 2.** $NECP_2(1,1,cat_1) = NRE$.

*Proof.* Let us consider a programmed grammar $G = (N, \{a\}, P, S)$ with appearance checking and with unconditional transfer. We add to $P$ the production $(0 : U \to S, \sigma(0), \varphi(0))$ with $\sigma(0) = \varphi(0) = l(S)$, where $U$ is a new non-terminal and $l(S)$ the set of labels of $S$-productions from $P$. In this way, we obtain a new grammar, $G' = (N', \{a\}, P', U)$, with $N' = N \cup \{U\}$. Clearly, $L(G) = L(G')$ and each derivation in $G'$ starts with the rule with label 0.

We construct the EC P system

$$\Pi = (O, \mu, w_1, w_2, R_1, R_2, R'_1, R'_2, i_o),$$

with

$$\begin{aligned}
O &= N' \cup \{d_i, d'_i, d''_i, e_i \mid i \in lab(P')\} \\
&\quad \cup \{a, F, F', F'', c, f, g, g', \#, h, p_1, p_2, \cdots, p_5\}, \\
\mu &= [_1[_2\,]_2]_1, \\
w_1 &= f, \\
w_2 &= cUhF, \\
i_o &= 0, \\
R_1 &= \{d_i \to d'_i, d''_i \to d'_i e_i \mid i \in lab(P')\} \\
&\quad \cup \{Y \to \# \mid Y \in N'\} \cup \{F' \to F'', \# \to \#\}, \\
R_2 &= \{F \to F, cF \to cF', g \to \#, cg \to cg', \# \to \#\} \\
&\quad \cup \{cX \to cxd_k g \mid (k : X \to x, \sigma(k), \varphi(k)) \in P'\} \\
&\quad \cup \{d_i \to \#, d''_i \to \#, ch \to chd''_i p_1 \mid i \in lab(P')\} \\
&\quad \cup \{cp_1 \to cp_2, cp_2 \to cp_3, cp_3 \to cp_4, cp_4 \to cp_5\}, \\
R'_1 &= \{(a, out)\},
\end{aligned}$$

$$R'_2 = \{(F', out), (a, out), (p_5, out), (d_0, out; f, in)\}$$
$$\cup \{(d_i, out; d'_j, in), (d''_i, out; d'_j, in) \mid i \in \sigma(j)\}$$
$$\cup \{(Y, out; F'', in) \mid Y \in N'\}$$
$$\cup \{(p_5, out; e_i, in), (Q, out; e_i, in) \mid (i : Q \to x, \sigma(i), \varphi(i)) \in P'\}.$$

The system $\Pi$ simulates the programmed grammar $G'$ in the following way. The evolution rules, of the form $cX \to cxd_kg$, for $(k : X \to x, \sigma(k), \varphi(k)) \in P'$, present in region 2, simulate the context-free rules of $G'$, while the symport/antiport rules are used to check if the simple evolution rules are applied according to the order defined by the labels in the success fields of $G'$. Moreover, the evolution rules of the form $ch \to chd''_ip_1$, for $i \in lab(P')$, present in region 2, are applied to skip a rule that cannot be applied and, in this case, the symport/antiport rules are used to be sure that the skipping of a rule has been done in a correct way (this mechanism simulates the appearance checking mechanism of the grammar $G'$). The evolution rules $F \to F, cF \to cF'$, together with the symport/antiport rules $(F', out), (Y, out; F'', in)$, for $Y \in N'$, are used to make sure that, when the computation halts, there is no symbol from $N'$ in region 2. The catalyst $c$ is used to ensure that in each step only one simple evolution rule is executed.

We pass now to show in more detail how a computation of $\Pi$ works. After a simple evolution (catalytic) rule of the form

$$cZ \to czd_mg, \text{ for some } (m : Z \to z, \sigma(m), \varphi(m)) \in P',$$

has been applied in region 2, the objects $d_m$ and $g$ have been produced. The object $d_m$ indicates that the rule of $G'$ with label $m$ has been applied, while the object $g$ is used only to "keep busy" the catalyst $c$. After the object $d_m$ has been produced, it must leave region 2 (because of the simple evolution rule that produces the trap symbol #). But the object $d_m$ can exit only if in region 1 the "right" predecessor ("right" in the sense of the order defined by the success/failure fields of $G'$) is present. In fact, to exit from region 2, the object $d_m$ must use (one of) the antiport rules from $R'_2$ of the form

$$(d_m, out; d'_j, in), \text{ for } j \text{ such that } m \in \varphi(j).$$

When the object $d_m$ goes in region 1, it is changed in $d'_m$ and this object will store the new predecessor (this means that it stores the label of the last applied rule) and the computation will continue. If a rule cannot be applied, then we must simulate the appearance checking mechanism of the grammar. It is possible to skip the application of a rule by using the simple evolution rule

$$ch \to chd''_qp_1, \text{ for some } q \in lab(P'),$$

present in region 2. The application of this rule means that the context-free rule of $G'$ with label $q$ cannot be applied. We obtain the objects $d''_q$ and $p_1$. The object $d''_q$ must go to region 1, but, as in the previous case, it can pass to region 1 only if the "right" predecessor is stored in this region. In fact, in order to exit

from region 2, the object $d''_q$ must use (one of) the antiport rules associated with membrane 2,

$$(d''_q, out; d'_j, in), \text{ for } j \text{ such that } q \in \varphi(j).$$

When the object $d''_q$ arrives in region 1, the evolution rule $d''_q \to d'_q e_q$, is applied, which creates the new predecessor $d'_q$ and the object $e_q$ that must check if the rule $(q : Q \to z, \sigma(q), \varphi(q))$ was skipped correctly.

After producing $e_q$, the antiport rule $(Q, out; e_q, in)$, associated with membrane 2 might be applied. If this happens, then the evolution rule $Q \to \#$, is applied in region 1 and the computation never halts (this means that the skipping of the rule with label $q$ was not correct).

If this antiport rule is not applied, then the skipping of the rule was correct and, now, the only thing to do is to "clean" region 1 of the object $e_q$. To this aim we use the object $p_5$ (it will arrive in region 1 only after checking the correctness of skipping a rule) and the antiport rule $(p_5, out; e_q, in)$, associated with membrane 2. After removing the object $e_q$ from region 1, the simulation of rules from $G'$ can be repeated. The objects $p_1, \cdots, p_5, g, g'$ are used to "keep busy" the catalyst $c$ and to send, at the right time, the object $p_5$ to region 1.

Each produced terminal is sent out by the symport rule associated to membranes 1, 2 and the system $\Pi$ sends out exactly the strings of $L(G) = L(G')$, hence, in this case it generates the length set of $L(G)$. □

## 5 Universality with Non-cooperative Rules

In this section we prove that EC P systems are universal even when using only simple non-cooperative evolution rules – hence no catalyst –, and simple symport/antiport rules. The "price" to pay, in comparison to the previous result, is to use a membrane structure composed of three membranes. The idea of the proof is to simulate a programmed grammar with appearance checking and unconditional transfer. As before, we use the fact that a programmed grammar with appearance checking and unconditional transfer is able to generate the recursively enumerable one-letter languages.

**Theorem 3.** $NECP_3(1, 1, ncoo) = NRE$.

*Proof.* Consider a programmed grammar $G = (N, \{a\}, P, S)$ with appearance checking and with unconditional transfer. As in the previous theorems, we add to $P$ the triple $(0 : U \to S, \sigma(0), \varphi(0))$ with $\sigma(0) = \varphi(0) = l(S)$ where $U$ is a new non-terminal and $l(S) = \{k \in lab(P) \mid (k : S \to x, \sigma(k), \varphi(k)) \in P\}$. In this way, we obtain a new grammar, $G' = (N', \{a\}, P', U)$, with $N' = N \cup \{U\}$; clearly, $L(G) = L(G')$ and each derivation in $G'$ starts with the rule with label 0.

For each $x \in N'$ we consider a new symbol $\bar{x}$, we denote by $\bar{N}$ the set of such symbols, and we define the morphism $\gamma : (N' \cup T)^* \longrightarrow (\bar{N} \cup T)^*$ by $\gamma(x) = \bar{x}$, if $x \in N'$, and $\gamma(x) = x$, if $x \in T$.

We construct the EC P system

$$\Pi = (O, \mu, w_1, w_2, w_3, R_1, R_2, R_3, R'_1, R'_2, R'_3, i_o),$$

with:

$$
\begin{aligned}
O &= N' \cup \bar{N} \cup \{d_i, d_i'', d_i''', e_i, e_i' \mid i \in lab(P')\} \\
&\quad \cup \{a, p_1 \cdots p_5, c, h, \bar{h}, c_1, c_2, \cdots, c_7, c', \#, F, F', f\}, \\
\mu &= [_1[_2[_3\ ]_3]_2]_1, \\
w_1 &= \emptyset, \\
w_2 &= FfUh, \\
w_3 &= c, \\
R_1 &= \{e_i \to e_i' \mid i \in lab(P')\} \cup \{Y \to \# \mid Y \in N'\}, \\
R_2 &= \{\bar{Y} \to Y \mid Y \in N'\} \cup \{\bar{h} \to h, F \to F, c \to c_1, c_1 \to c_2, \cdots, c_6 \to c_7\} \\
&\quad \cup \{d_i \to d_i', d_i'' \to d_i' e_i \mid i \in lab(P')\}, \\
R_3 &= \{X \to \gamma(x) d_k \mid (k : X \to x, \sigma(k), \varphi(k)) \in P'\} \\
&\quad \cup \{h \to \bar{h} d_i'' p_1, d_i \to \#, d_i'' \to \# \mid i \in lab(P')\} \\
&\quad \cup \{p_1 \to p_2, p_2 \to p_3, \cdots, p_4 \to p_5, c_7 \to c, c \to c', F \to F'\}, \\
R_1' &= \{(a, out)\}, \\
R_2' &= \{(Q, out; e_i', in), (e_i, out), (p_5, out; e_i', in) \mid (i : Q \to z, \sigma(i), \varphi(i)) \in P'\} \\
&\quad \cup \{(F', out), (a, out)\} \cup \{(Y, out; F', in) \mid Y \in N'\}, \\
R_3' &= \{(\bar{Y}, out), (c, out; Y, in) \mid Y \in N'\} \\
&\quad \cup \{(\bar{h}, out), (F', out), (c', out; F, in), (c, out; h, in), (c_7, in), (a, out)\} \\
&\quad \cup \{(d_i, out; d_j', in), (d_i'', out; d_j', in) \mid i \in \sigma(j)\}.
\end{aligned}
$$

The system $\Pi$ simulates the programmed grammar $G'$ in the following way. In region 3 (the inner one) we simulate the application of the context-free rules of $G'$, using the simple evolution rules

$$X \to \gamma(x) d_k, \text{ for } (k : X \to x, \sigma(k), \varphi(k)) \in P',$$

while the symport/antiport rules

$$(d_i, out; d_j', in), (d_i'', out; d_j', in), \text{ for } i \text{ and } j \text{ such that } i \in \sigma(j),$$

associated with membrane 3, take care of the application in the correct order of the rules of $G'$.

During the simulation, using the symport/antiport rules

$$(c, out; Y, in), \text{ for } Y \in N',$$

associated with membrane 3, we move from region 2 to region 3 the non-terminal that must be rewritten, and after performing such a rewriting, we store in region 2 the objects obtained. Finally, we use region 1 to implement the appearance checking mechanism of $G'$, using the antiport rules

$$(Q, out; e_i', in), \text{ for } (i : Q \to z, \sigma(i), \varphi(i)) \in P',$$

associated with membrane 2.

The evolution rules $F \to F, F \to F'$, present in region 2 and 3 respectively, together with the antiport rule $(c', out; F, in)$ associated with membrane 3, and with the symport rule $(F', out)$ associated with membranes 3 and 2, are used to make sure that, when the computation halts, there are no objects from $N'$ in region 2.

Now we pass to describe in more details the work of the system $\Pi$.

After applying an antiport rule

$$(c, out; Z, in), \text{ for some } Z \in N',$$

associated with membrane 3, the object $Z$ arrives in region 3 and the evolution rule

$$Z \to \gamma(z)d_m, \text{ with } (m : Z \to z, \sigma(m), \varphi(m)) \in P',$$

is applied. The non-terminal objects of $\gamma(z)$ are sent to region 2, using the symport rules

$$(\bar{Y}, out), \text{ for } Y \in N',$$

associated with membrane 3. The object $d_m$ indicates that the rule of $G'$ with label $m$ has been applied; as soon as such an object has been produced it must leave region 3 (because of the evolution rule that produces the trap symbol #). The object $d_m$ can exit only if in region 2 there exists the "right" predecessor ("right" in the sense of the order defined by the success/failure fields of $G'$). In fact, to leave region 3 the object $d_m$ must use (one of) the antiport rules associated with membrane 3 of the form

$$(d_m, out; d'_j, in), \text{ for } m \in \sigma(j).$$

Such an antiport rule can be applied only if in region 2 one of the predecessors of $d_m$ is present. When the object $d_m$ goes in region 2, it is changed in $d'_m$ and this object will store the new predecessor and the computation continues.

If a rule cannot be applied, then we must simulate the appearance checking mechanism of the grammar: we introduce in region 3 the special object $h$ using the antiport rule $(c, out; h, in)$, associated with membrane 3, and, after that, we can apply the simple evolution rule

$$h \to \bar{h}d''_q p_1, \text{ for some } q \in lab(P'),$$

present in region 3. The application of this rule means that the context-free rule of $G'$ with label $q$ cannot be applied. The application of this evolution rule produces the objects $d''_q$, $\bar{h}$, and $p_1$. The object $\bar{h}$ is immediately sent out to region 2 and the object $d''_q$ must go to region 2, but, as in the previous case, it can pass to region 2 only if the "right" predecessor is there. In fact, to exit from region 3, the object $d''_q$ must use (one of) the antiport rules associated with membrane 3

$$(d''_q, out; d'_j, in), \text{ for } q \in \sigma(j).$$

In this case, when the object $d''_q$ arrives in region 2, the evolution rule $d''_q \to d'_q e_q$, is applied and it creates the new predecessor $d'_q$ and the object $e_q$ that will check

if the rule with label $q$ was skipped correctly. After producing $e_q$, this object is sent out to region 1, using the symport rule $(e_q, out)$, associated with membrane 2. As soon as the object $e_q$ arrives in region 1, it is changed to $e'_q$ and, at the next step, the antiport rule

$$(Q, out; e'_q, in), \text{ for } (q : Q \to z, \sigma(q), \varphi(q)) \in P',$$

associated with membrane 2, might be applied. If this happens, then the computation never halts, because the evolution rule $Q \to \#$, present in region 1, is applied (this means that the skipping of the rule with label $q$ was not correct). If this antiport rule is not applied, then the skipping of the rule was correct and, now, the only thing to do is the cleaning of region 1 of the object $e'_q$. To this aim we use the object $p_5$ (it will arrive in region 2 only after checking the correctness of skipping a rule), together with the antiport rule

$$(p_5, out; e'_q, in),$$

associated with membrane 2. After removing the object $e'_q$ from region 1, the simulation of the rules from $G'$ can be repeated. Finally, we observe that the objects $p_1, p_2, \cdots, p_5$ are used to send at the "right" time the object $p_5$ to region 1, and the objects $c_1, c_2, \cdots, c_7$ are used to "keep busy" the special object $c$.

In each step, each produced terminal is sent out by the symport rules associated with membrane 1, 2, and 3, and we collect the output in the enviroment. Thus, the system $\Pi$ sends out exactly the strings of $L(G) = L(G')$, and, hence, it generates the length set of $L(G)$. □

## 6 Open Problems

We have studied here the evolution-communication P systems with what we called the *mixed approach*, which is only one (probably the less restrictive) among the possible ways to define a computation in an EC P system. We can have, at least, two other approaches that we describe here, rather informally; these two approaches might be seen as inspired, in some extent, from biology:

(i) the *evolutive approach*:
the simple evolution rules of the system have priority over the symport/antiport rules (in *biological* terms: an organism evolves by itself as much as possible (*evolution*), until it needs something from others organisms (*communication*));

(ii) the *communicative approach*:
the symport/antiport rules of the system have priority over the simple evolution rules (in *biological* terms: an organism tries to *communicate* with the others organisms, and only when it cannot comunicate anymore, then it starts (or continue) its *evolution*).

We consider interesting the study of these (and possibly other) different approaches to have computation in an EC P system. For instance, a general question of interest could be to check what is *more important*, in this framework, evolution or communication.

Moreover, we want point to out that we have given two universality results for the EC P systems proving the equivalence with $NRE$. What can we say about the equivalence with $PsRE$? Are similar results true?

Finally, we consider of interest to study the relations between the power of simple evolution rules and the power of symport/antiport rules. In other words, is there some kind of trade-off between these two types of rules?

## 7 Final Remarks

We have introduced a new variant of P systems called evolution-communication P system, that simply joins classical evolution rules (without communication targets) and communication rules (symport/antiport rules). This new model seems to be closer to the biology, because it does not use target indications, but simple symport/antiport rules to realize the communication. We have presented a particular way of defining a computation in such a system, called the *mixed approach*, and we have proven that the EC P systems are universal when using catalytic rules (with only one catalyst), and symport/antiport rules of weight one, with a membrane structure consisting of two membranes; moreover, we have proven that, if we use three membranes, we can have universality also without catalysts (hence with non-cooperative rules) and, again, using symport/antiport rules of weight one. We can say that the results presented in this paper can stay in the middle between the results presented in [5] and [6] on P systems with evolution rules (or catalytic rules) and with target indications, and the results presented in [4] on P systems with symport/antiport rules. Finally we have suggested some different strategies (*evolutive and communicative*) to define a computation in this new variant; also, several open problems are presented.

## References

1. F. Bernardini, V. Manca, P systems with boundary rules, *Pre-proceedings of Workshop on Membrane Computing, WMC-CdeA2002* (G. Păun, C. Zandron, eds.), Curtea de Argeş, Romania, August 2002, 97–101.
2. J. Dassow, G. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
3. H. Fernau, F. Stephan: How powerful is unconditional transfer? When UT meets AC, *Proceedings of the 3rd International Conference on Developments in Language Theory* (S. Bozapalidis, ed.), 1997, 249–260.
4. P. Frisco, H.J. Hoogeboom, Simulating counter automata by P systems with symport/antiport, *Pre-proceedings of Workshop on Membrane Computing, WMC-CdeA2002* (G. Păun, C. Zandron, eds.), Curtea de Argeş, Romania, August 2002, 237–248.
5. G. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, Heidelberg, 2002.
6. P. Sosik, P systems versus register machines: two universality proofs, *Pre-proceedings of Workshop on Membrane Computing, WMC-CdeA2002* (G. Păun, C. Zandron, eds.), Curtea de Argeş, Romania, August 2002, 371–382.

# Dynamic P Systems

Rodica Ceterchi[1,⋆] and Carlos Martín-Vide[2]

[1] Faculty of Mathematics, University of Bucharest
14, Academiei st., 70109 Bucharest, Romania
rc@funinf.math.unibuc.ro
[2] Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tárraco 1, 43005 Tarragona, Spain
cmv@astor.urv.es

**Abstract.** We propose in this paper a uniform manner of generating families of P systems, using contextual grammars, specifically, variants of bracketed contextual grammars. We introduce the concept of enriched bracketed contextual grammar, which can be used to generate families of P systems with symport/antiport rules.
More generally, we define the notion of dynamic P system associated to grammars able to "generate" families of P systems, and introduce the notion of dynamic computation sequence, which combines the previous generative approach with computational aspects.
We illustrate these notions with an application to the problem of sorting. The model of sorting we propose uses symport/antiport rules with priority relations, and the notion of stable configuration of a P system.

## 1 Introduction

P systems are a very powerful computational tool, recently introduced by G. Păun in [10]. Several variants were proposed and studied, and the literature on the subject is growing rapidly. The main focus is the investigation of the computational power of P systems.

In this paper we step for a moment aside from the main trend, and ask the question if it is possible to *generate families of P systems in a coherent, uniform way*. The observation which has started this research is that the membrane structure of very simple P systems, with only one symport/antiport rule per membrane, and no strings, is described by a certain type of bracketed strings. *Languages* of bracketed strings are generated by a certain class of contextual grammars, the bracketed and fully bracketed contextual grammars, introduced in [7]. A derivation in such a grammar will thus transform a P system into another, more complex one.

We can thus conceive of finding *generative mechanisms for classes of P systems*, in the form of (new types of) contextual grammars, whose derivations

⋆ Work supported by the grant SAB2000-0145, the Secretaría de Estado de Educación y Universidades, Spanish Ministry for Education, Culture and Sport.

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 146–186, 2003.

act on P systems, producing new P systems. This is the first main topic of the present paper. Once we have such a generative mechanism – or, better – a recipe for constructing such mechanisms, we will have a new, more precise meaning attached to "family", or "class" of P systems, and we will be able to address new types of problems, such as measures of "resemblance/similarity" between P systems, the descriptional complexity of such classes of P systems, and, last but not least, the computational power of such classes.

The second main topic of the present paper is to combine the generative approach, with the computational one. The very general concept of dynamic P system, and its associated dynamic computation sequence, captures (some of) the spirit of an "evolving-while-computing" P system: the *coherent changes*, which can be thought of as evolution steps, can now be modelled by the generative devices in the form of grammars, proposed by the previous generative approach; after each evolution step, the system is able to perform new computations, and these, in turn, create the conditions for the next evolution step.

We illustrate how the concept of dynamic P system can be used to solve *dynamic problems*, by applying it to the problem of sorting. The model for sorting which we propose here, and which is based on P systems with communication rules, can be considered as a third main topic, independent of the previous ones. We believe it deserves further, and independent, investigation. We present here only some dynamic versions, as illustrations for the concept of dynamic P system.

The paper is organized as follows.

Section 3 is devoted to "the generative approach". It is based on research started in [3]. We first recall the bracketed and fully bracketed contextual grammars, and we establish their connection with P systems. Then, in subsections 3.3 and 3.4, we introduce new types of bracketed strings, and evaluate new types of contextual productions from the point of view of what kind of P systems they are able to generate. In subsection 3.5 we introduce the formal definitions of enriched bracketed contextual grammars, which are generative devices for P systems with symport/antiport rules, and symbol-objects. Even though they are initially introduced for P systems as above with totally ordered sets of rules, we show in subsection 3.6 that the same concept applies to P systems with communication in the most general sense. We also point in this section to the generality of the generative approach (other kinds of P systems, and other kinds of contextual grammars could be considered), and to some questions which can be asked in this new area.

In section 4 we introduce the notion of dynamic P system and its associated dynamic computation sequence. We introduce it in a very general setting, in order to make it independent of the specifics of the present paper. The fact that it can be used in other settings, and is independent of the concept of enriched bracketed contextual grammar for instance, has been proved by the application developed in [2].

Section 5 is intended to illustrate how the concept of dynamic P system and its associated dynamic computation sequence can be used to deal with a *dynamic problem*; the problem considered here is that of sorting. In subsection 5.1 we

present the main traits of our model for sorting (a model whose main idea is *not* related to the other two main topics of this paper). In subsection 5.3 we present the basic mechanism, the comparator, implemented using communication rules with priority relations. In subsection 5.4 we present a first dynamic P system which solves the sorting problem, by using an enriched bracketed $n$-contextual grammar. A second dynamic P system which solves the same problem, based this time on an enriched bracketed insertion grammar, is presented in subsection 5.5. Subsection 5.6 is devoted to comparing our approach to sorting, to the approach in [1], which also uses symport/antiport rules.

Finally, section 6 is devoted to concluding remarks, and open problems.

Section 2, with the inconspicuous title "Preliminaries", contains what it advertises, i.e., some basic definitions of contextual grammars (2.2), and of P systems with communication (2.1), but also something more. In subsection 2.2, some slight extensions of classical notions – deterministic contextual grammars, contextual grammars with infinite sets of contexts, insertion grammars with infinite sets of insertion strings – are presented, in order to be used (but not in a very essential way) in subsections 5.4 and 5.5. In subsection 2.1 we define the notion of stable configuration of a P system, which is mentioned in section 4, and used as a basic ingredient for our model of sorting in section 5.

## 2 Preliminaries

### 2.1 P Systems with Communication

In our paper we will focus only on P systems with communication, i.e., systems which compute by moving symbol or string objects between adjacent membranes. Let us note that communication makes the difference between a collection of separate computing agents and a *system* of cooperating agents. Furthermore, in [8] it is shown that *communication alone is capable of universal computations*, which shows that focusing on systems which have *only* communication is not devoid of interest.

The main notations and definitions in the rest of this section can be found in [11], with the exception of some new concepts which we will introduce.

By communication we will mean communication/transmission of objects between membranes, obviously between adjacent membranes. We will associate to membranes two types of communication rules:

– *symport rules* are of the type $(x, in)$, with the meaning "objects described by the string $x$ can enter (the membrane to which the rule is associated)", and $(x, out)$, with the meaning "objects described by the string $x$ can exit (the membrane to which the rule is associated)";

– *antiport rules* are of the type $(x, in; y, out)$, with the meaning "objects described by the string $x$ enter, while objects described by $y$ exit the membrane".

Note that antiport rules with one argument $\lambda$ become symport rules, for instance, $(x, in; \lambda, out) = (x, in)$.

**Definition 1.** *A* P system with symport/antiport *is a construct*

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

*where*

*(i) $O$ is the alphabet of objects;*

*(ii) $\mu$ is a membrane structure with $m$ membranes;*

*(iii) strings $w_i$, $1 \leq i \leq m$, represent the multisets over $O$ associated with the regions of $\mu$;*

*(iv) $E \subseteq O$ is the set of objects which are supposed to appear in the environment in arbitrarily many copies;*

*(v) $R_i$, $1 \leq i \leq m$, are finite sets of symport and antiport rules over $O$ associated with the membranes of $\mu$;*

*(vi) $i_0$ is the label of an elementary membrane of $\mu$ (the output membrane).*

In what follows, two components of such a construct will be ignored, the environment, $E$, and the output membrane, $i_0$. When referring to such a system we will say that it is "not fully specified".

The membrane structure $\mu$ of a P system will play an essential role in our approach, and we recall that it is described by a word in the Dyck language over the alphabet $\{[,]\}$ of pairs of brackets.

Also, we mention the fact that in general, in the P systems literature, we have not (yet) encountered the idea of sets of symport/antiport rules endowed with priority relations, that is, with a partial order relation, which plays a role in the way rules are applied. Priority relations on rules have been used for P systems with (multiset or string) rewriting rules. We use the notion of priority *not* in its strong sense, but in the weak sense, of *competition for objects*.

Also of biochemical inspiration is the idea of "promoters/inhibitors" of rules: some rules are active only in the presence of certain objects – the promoters, and become de-activated in the presence of other objects – the inhibitors. References can be found in [11] and [13].

For our purposes, in the second part of the paper, we will need a very simple version, symport and antiport rules with one promoter, which can be a symbol-object in $O$.

Let $p \in O$ be a distiguished symbol-object, our promoter. We denote by $(x, in; y, out)|_p$ the antiport rule $(x, in; y, out)$ which will be active in a membrane only if object $p$ is present in that membrane, and not active if $p$ is absent. Similarly, for symport rules, we denote $(x, in)|_p$ the symport rule $(x, in)$, active if and only if $p$ is present in the membrane.

Promoters themselves can be the object of comunication rules. In order to avoid the cumbersome notation $(p, out)|_p$ which has the meaning "if $p$ is present then $p$ gets out of the membrane", we use $(p, out)$, which has the same meaning, and acts in the same way (if $p$ is not present, the rule cannot be applied).

We will assume that, if a promoter is present in a membrane where it can activate some rules, and it is also subject to rules which can take it out of that membrane, then it will first activate the rules in question, and only afterwards exit the membrane.

By a *configuration* of a P system we will mean a state of the system described by specifying the membrane structure, together with the objects and rules associated to each membrane.

The notion of *a halting computation* is well known in the P systems literature: after all possible application of the rules, when the system has reached a configuration in which no rule is any longer applicable, we say that the computation halts. In standard P systems literature, a *successful* computation is a halting one.

For the second part of our paper we will need a weaker notion than that of a successful computation, which does not imply halting.

We say that a P system has reached *a stable configuration* if, even if some rules are still applicable, their application does not change the string/object content of the membrane structure, nor the membrane structure itself.

Two very easy examples of rules whose action does not change the content of the membranes of a P system are the following:

1. An antiport rule of the type $(x, in; x, out)$. If only this rule is active, in the inner membrane of a system containing two nested membranes, then the P system has reached a stable configuration, since the number of occurrences of $x$ in the two regions remains the same, and the rest is unchanged.

2. Pairs of rewriting rules of the type $x \longrightarrow y$ and $y \longrightarrow x$. If in a membrane we have the same number of occurrences of $x$ and $y$, then the application of these rules does not change the content of the membrane, the P system is stable.

Even if, at the first sight, these rules might seem absurd, or useless, in a given context they might accomplish useful functions. We hope to prove that this is the case with the first rule, which will be used in subsection 5.3 to accomplish the construction of a comparator. We have not found yet something similar for the second type of rules. Note that, for instance in [5], the first type of rule is used to *mark* unsuccessful computations.

We think also of the fact that, in biological entities, keeping a balance for some chemical substances, or electrical charges, is not an uncommon feature. These rules seem to be, maybe in a very rough form, the formal expression for such equilibrium mechanisms.

Note that this notion of stability is similar to the notions of *adult* word, and *adult* languages, from the L systems area.

We will also consider the following weaker notions of stability.

If $W \subseteq O$ is a sub-alphabet of the alphabet of objects, then we call a P system *stable w.r.t.* $W$ if the projection over $W$ of the string/object content of the system's membranes remains unchanged, even if some rules are still applicable.

We have here in mind the fact that not all changes in a P system are of equal importance, and that we might be interested in (or might be able to observe) only some of its objects. Note that stable w.r.t. $O$ is the same as stable.

If $R' = (R'_1, \cdots, R'_m)$, with $R'_i \subset R_i$ for $1 \leq i \leq m$, is a subset of the rules of a P system, we call a P system *stable w.r.t. the rules* $R'$ if the P system with rules $R'$ is stable (i.e., applications of rules in $R'$ do not change the string/object content of the system's membranes).

We will use this second notion of stability to formulate more precisely the above mentioned behavior of promoters.

Suppose a promoter $p$ is present in a membrane $i$, where it can activate the subset of rules ${R_i}^p$.

*The promoter p cannot leave membrane i, unless the system has reached a configuration which is stable w.r.t. the rules ${R_i}^p$ activated by p.*

Also, the priority relations between communication rules can now be endowed with a more precise meaning. If $r_1 > \cdots > r_n > r_{n+1} > \cdots$ is a totally ordered subset of communication rules inside a membrane, then *rule $r_{n+1}$ is applicable iff the system has reached a configuration which is stable w.r.t. rules $r_1 > \cdots > r_n$.*

As an example, consider a system with two nested membranes, $[_2[_1]_1]_2$, with $R_2 = \emptyset$, and membrane 1 having the set of rules

$$R_1 = (a, in; a, out)|_p > (p, out),$$

and symbol-objects $w_1 = a^n p$, $w_2 = a^n$.

We recall that our priority relation on rules acts in a "competing for objects" manner. Then, the first rule will be active because of the presence of $p$, and it will exchange the $n$ occurrences of $a$ from membrane 1 with the $n$ occurrences of $a$ from membrane 2. Since the number of occurrences of $a$ in each membrane does not change, the system is stable w.r.t. the rule $= (a, in; a, out)|_p$, and the second rule is now applicable, and will get $p$ out from membrane 1 into 2. The content of the membranes will be now $w_1 = a^n$ and $w_2 = a^n p$.

### 2.2 Contextual Grammars

We recall from [9] the following concepts.

A *contextual grammar (with choice)* is a construct $G = (V, A, C, \phi)$, where $V$ is an alphabet, $A$ is a finite language over $V$, called the set of *axioms*, $C$ is a finite subset of $V^* \times V^*$, called the set of *contexts*, and $\phi : V^* \to \mathcal{P}(C)$ is the *choice (or selection) function* of the grammar $G$.

With respect to a contextual grammar with choice, we can define two derivation relations:

$$x \Longrightarrow_{ex} y \text{ iff } y = uxv, \text{ for some } (u, v) \in \phi(x),$$
$$x \Longrightarrow_{in} y \text{ iff } x = x_1x_2x_3 \text{ and } y = x_1ux_2vx_3, \text{ for some } (u, v) \in \phi(x_2).$$

We say that $\Longrightarrow_{ex}$ is an *external* derivation, whereas $\Longrightarrow_{in}$ is an *internal* derivation in $G$. If $\Longrightarrow^*_{ex}$ and $\Longrightarrow^*_{in}$ are the reflexive and transitive closures of $\Longrightarrow_{ex}$, respectively $\Longrightarrow_{in}$, then $L_\alpha(G) = \{x \in V^* \mid w \Longrightarrow^*_\alpha x \text{ for some } w \in A\}$ denotes the language generated by $G$ in each one of the two cases ($\alpha \in \{ex, in\}$).

We say that such a grammar, used in the external mode is an *external* contextual grammar (with choice), whereas used in the internal mode it is an *internal* contextual grammar (with choice).

We say that such a grammar is an (external, respectively internal) contextual grammar *without choice* iff $\phi(x) = C$, for all $x \in V^*$. In this case, the derivation

styles can be rewritten as:

$$x \Longrightarrow_{ex} y \text{ iff } y = uxv, \text{ for some } (u, v) \in C,$$
$$x \Longrightarrow_{in} y \text{ iff } x = x_1x_2x_3 \text{ and } y = x_1ux_2vx_3, \text{ for some } (u, v) \in C.$$

A contextual grammar $G = (V, A, C, \phi)$ is in the *modular presentation* when it is given as a construct $G = (V, A, (S_1, C_1), \ldots, (S_n, C_n))$, such that $C_i \subseteq \phi(x)$, for all $x \in S_i$, $1 \leq i \leq n$. A pair $(S_i, C_i)$ is called a *production*, with $S_i$ the *selector* of the production, and $C_i$ its *contexts*.

An *n-contextual grammar* is a construct $G = (V, A, C, \phi)$, where $V$ is an alphabet, $A$ is a finite language over $V$ (the set of *axioms*), $C$ is a finite subset of $(V^*)^n$ (the set of *n-contexts*), and $\phi : (V^*)^{n+1} \to \mathcal{P}(C)$ is the *choice (or selection) map*. The derivation in an $n$-contextual grammar is defined as:

$$x \Longrightarrow y \text{ iff } x = x_1x_2 \ldots x_{n+1} \text{ and } y = x_1u_1x_2u_2 \ldots x_nu_nx_{n+1}$$
$$\text{for } x_1, x_2, \ldots, x_{n+1} \in V^* \text{ and } (u_1, u_2, \ldots, u_n) \in \phi(x_1, x_2, \ldots, x_{n+1}).$$

If $\Longrightarrow^*$ is the reflexive and transitive closure of $\Longrightarrow$, then $L(G) = \{x \in V^* \mid w \Longrightarrow^* x \text{ for some } w \in A\}$ denotes the language generated by $G$.

A 2-contextual grammar is also called *total contextual grammar*.

An *insertion grammar* is a construct $G = (V, A, P)$, where $V$ is an alphabet, $A$ is a finite language over $V$ (the set of *axioms*), and $P$ is a finite subset of $V^* \times V^* \times V^*$ called the set of *insertion rules*. The derivation in an insertion grammar is defined as:

$$x \Longrightarrow y \text{ iff } x = x_1uvx_2 \text{ and } y = x_1uzvx_2, \text{ for some } (u, z, v) \in P.$$

If $\Longrightarrow^*$ is the reflexive and transitive closure of $\Longrightarrow$, then $L(G) = \{x \in V^* \mid w \Longrightarrow^* x \text{ for some } w \in A\}$ denotes the language generated by $G$.

Recall also from [9], that the following extension was proposed and studied for internal contextual grammars with choice, the contextul grammars *with an infinite set of contexts*. The definition can be extended along the same lines to several of the above types.

More precisely, let the construct $G = (V, A, C, \phi)$ denote an internal/external contextual grammar with choice, or an $n$-contextual grammar. We say that $G$ is a contextual grammar (of the respective type) *with an infinite set of contexts* iff:

– the total set of contexts, $C$, is allowed to be infinite;

– the image $\phi(x)$ of the selection function, in every point $x$ of its domain, is finite.

In other words, even if we allow for infinity of the total set o contexts, the selection mechanism (and thus the derivations) should act in a finite manner.

Also, the concept of *deterministic* contextual grammars was considered in [9], actually only for grammars with insertion and deletion of contexts.

We can extend this concept in a natural way to several of the above mentioned types of contextual grammars.

Let the construct $G = (V, A, C, \phi)$ denote a contextual grammar of any of the above types. We say that $G$ is a *deterministic* contextual grammar (of the respective type) iff $|\phi(x)| \leq 1$ for every $x$ in the domain of $\phi$.

Note that, for the above mentioned types, we can now combine the two features, and consider *deterministic contextual grammars with an infinite set of contexts*.

We will use the concept of deterministic $n$-contextual grammar, with a possibly infinite set of contexts, in subsection 5.4.

For insertion grammars, we propose the following extensions, which, we believe, capture the same spirit in a different formal setting.

Let $G = (V, A, P)$, be an insertion grammar. Let $C = \{(u, v) \mid (u, x, v) \in P\}$, be its set of *contexts*, and $X = \{x \mid (u, x, v) \in P\}$, its set of *insertion strings*. Denote by

$$P_{(u,v)} = \{(u, x, v) \in P \mid x \in X\}$$

the set of insertion rules which use the context $(u, v)$.

We call $G = (V, A, P)$ an insertion grammar *with infinite set of insertion strings* if:

– $V$ and $A$ retain their meaning from the original definition;
– $X$ can be infinite, and thus $P$ can be infinite;
– for each context $(u, v) \in C$ we have $|P_{(u,v)}| < \infty$.

In other words, even if the set of all insertion strings is infinite, a given context can select only a finite number of strings to be inserted.

We call an insertion grammar $G = (V, A, P)$ (with finite or infinite set of insertion strings) *deterministic*, iff, for every context $(u, v) \in C$, we have $|P_{(u,v)}| \leq 1$.

We will refer to the concept of deterministic insertion grammar, with a possibly infinite set of insertion strings, in subsection 5.5.

## 3 From Contextual Grammars to P Systems – A Generative Approach

### 3.1 Bracketed Strings and Bracketed Contextual Grammars

Let $V$ be an alphabet and $B = \{[, ]\}$ the alphabet with one pair of parentheses. By $D_B$ we will denote the Dyck language over $B$.

*Remark 1.* Throughout the rest of this paper, we will use indices for the brackets only for the purpose of marking pairs of matching brackets. We will make explicit use of sets of separators such as $B_n = \{[_i, ]_i \mid 1 \leq i \leq n\}$, but the indices will not represent types, and the underlying Dyck language will be the Dyck language with one type of brackets.

Recall from [7] and [9] the following concepts:

**Definition 2.** *A string $x \in (V \cup B)^*$ is called a* Dyck covered string *iff using only reduction rules of type $[w] \to \lambda$ for any $w \in V^*$, we have $x \Longrightarrow^* \lambda$.*

We denote by $DC(V)$ the set of all Dyck covered strings over $V$ as above.

**Definition 3.** *A Dyck covered string $x \in (V \cup B)^*$ is called a* minimally *Dyck covered string iff:*

1. *if $x = x_1]x_2[x_3$ with $x_1, x_3 \in (V \cup B)^*$, $x_2 \in V^*$, then $x_2 = \lambda$.*
2. *the rule $[\,] \to \lambda$ is not used in the reduction $x \Longrightarrow^* \lambda$.*

We denote by $MDC(V)$ the set of all minimally Dyck covered strings over $V$.

**Definition 4.** *A string $x \in (V \cup B)^*$ is called a* Dyck covered string with skin *iff $x = [y]$, with $y \in DC(V)$, and* minimally Dyck covered with skin *iff $x = [y]$, with $y \in MDC(V)$.*

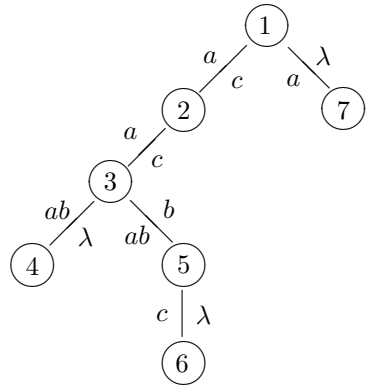



**Fig. 1**

We denote by $SDC(V)$, respectively $SMDC(V)$, the set of Dyck covered, respectively minimally Dyck covered strings with skin over $V$.

Recall from [7] and [9] that to each string in $MDC(V)$ we can associate in a unique manner a tree, with nodes labeled with integers, and doubly labelled edges with labels in $V^*$. The construction of the tree $\tau(x)$ from the string $x$ is as follows:

– draw the root node, and label it with 1;

– scan $x$ from left to right and grow $\tau(x)$ according to the rules:

(down) for each maximal $[w$, with $w \in V^*$, draw a new edge starting in the current node, place it to the right of it, and mark $w$ on the left side of the edge;

(up) for each maximal $w]$, $w \in V^*$, not scanned yet, climb the current edge, marking $w$ on the right side of it.

*Example 1.* Consider the string $[_2\ a\,[_3\ a\,[_4\ ab\,]_4\ [_5\ ab\ [_6\ c\ ]_6\ b\ ]_5\ c\ ]_3\ c\ ]_2\ [_7\ a\ ]_7$. We will associate to it the tree in Fig. 1. Note that the root node, with label 1, does *not* correspond to a pair of parentheses in the expression.

In [7] and [9], the following types of contextual grammars were considered.

Let $G = (V, A, P)$ be a contextual grammar in modular presentation, with

$$P = \{(S_i, C_i) \mid 1 \leq i \leq n\}$$

its set of productions.

**Definition 5.** *A contextual grammar* $G = (V, A, P)$ *(in modular presentation) is called a* bracketed contextual grammar *iff* $A \subseteq MDC(V)$, *and* $S_i \subseteq V^*$, $C_i \subseteq V^* \times V^* \setminus (\lambda, \lambda)$ *for all* $1 \leq i \leq n$.

Note that the axioms are strings in $MDC(V)$, while the selectors are strings in $V^*$, as for ordinary contextual grammars.

The derivation relation (in the internal mode) associated to a bracketed contextual grammar is defined as follows: for $x, y \in (V \cup B)^*$

$x \Longrightarrow_{in} y$ iff $x = x_1x_2x_3$ , $y = x_1[ux_2v]x_3$,
with $x_1, x_3 \in (V \cup B)^*$, $x_2 \in MDC(V)$, and
$pr_V(x_2) \in S_i$, $(u, v) \in C_i$, for some $i$,

where $pr_V$ denotes the projection on $V$, $pr_V : (V \cup B)^* \longrightarrow V^*$, defined as usual: $pr_V([) = pr_V(]) = \lambda$, and $pr_V(a) = a$ for $a \in V$.

**Definition 6.** *A contextual grammar* $G = (V, A, P)$ *is called a* fully bracketed contextual grammar, *iff* $A \subseteq MDC(V)$, *and* $S_i \subseteq MDC(V)$, $C_i \subseteq V^* \times V^* \setminus (\lambda, \lambda)$ *for all* $1 \leq i \leq n$.

For $x, y \in MDC(V)$ we define the derivation in a fully bracketed contextual grammar:

$x \Longrightarrow_{in} y$ iff $x = x_1x_2x_3$ , $y = x_1[ux_2v]x_3$,
with $x_1, x_3 \in (V \cup B)^*$, $x_2 \in MDC(V)$, and
$x_2 \in S_i$, $(u, v) \in C_i$, for some $i$.

Note that in the case of bracketed contextual grammars the selection is made by strings over $V$, embedded in brackets to give $x_2 \in MDC(V)$, while in the case of fully bracketed contextual grammars the selectors themselves have brackets, and are in $MDC(V)$.

The *string language* generated by a bracketed contextual grammar $G = (V, A, P)$ is defined by

$$L_{in}(G) = \{pr_V(w) \mid z \Longrightarrow_{in}^* w, \text{ for some } z \in A\}.$$

The *bracketed language* generated by a bracketed contextual grammar is defined by

$$BL_{in}(G) = \{(pr_V(w), \tau(w)) \mid z \Longrightarrow_{in}^* w, \text{ for some } z \in A\}.$$

Note that we can construct, from the given set of contexts, the alternative set $CC_i = \{([u, v]) \mid (u, v) \in C_i\}$, and then, the above derivations are the usual internal derivations in a contextual grammar.

Note also that, if we choose to work with axioms in $SMDC(V)$, then the above derivation relations produce strings still in $SMDC(V)$. The example bellow is of this type.

*Example 2.* Take the bracketed (not fully bracketed) contextual grammar

$$G = (\{a, b, c\}, \{[\ [ab]\ [c]\ [a]\ ]\}, \{p_1 = (c, ([ab, b])), p_2 = (abc, ([a, c]))\}\ ).$$

The string in Example 1 is obtained in this grammar by applying to the axiom twice the rule $p_2$, followed by one application of rule $p_1$. (Note that after applying $p_1$, rule $p_2$ cannot be applied any more.) The derivation is the following:

$$\begin{aligned}[ \, [ab] \, [c] \, [a] \, ] &\Longrightarrow_{p2} [ \, [ \, a \, [ab] \, [c] \, c \, ] \, [a] \, ] \\ &\Longrightarrow_{p2} [ \, [ \, a \, [ \, a \, [ab] \, [c] \, c \, ] \, c \, ] \, [a] \, ] \\ &\Longrightarrow_{p1} [ \, [ \, a \, [ \, a \, [ab] \, [ \, ab \, [c] \, b \, ] \, c \, ] \, c \, ] \, [a] \, ].\end{aligned}$$

### 3.2 The Crucial Observation

To every tree, $\tau(x)$, constructed as in Example 1, we can associate in a unique manner a membrane structure with symport/antiport rules, in the following way:

– to every node, labelled with the integer $i$, we associate a membrane $[_i \, ]_i$, and the root node, labelled with 1 will be associated to the skin membrane, $[_1 \, ]_1$;

– to every internal node, which has one (and only one) edge entering into it, with labels $(x, y)$, where $x$ is the "left" label and $y$ the "right" one, we associate the antiport rule $(x, in; y, out)$ with the membrane representing that node.

Note that all the *leaves* of such a tree will correspond to *elementary membranes*, and, having only labels of type $(w, \lambda)$, the rules for the elementary membranes will be symport rules of type $(w, in)$.

We have thus the following result:

**Lemma 1.** *Every string $x \in SMDC(V)$ describes a P system*

$$P_x = (V, \mu, w_1, \cdots, w_m, E, R_1, \cdots, R_m, i_0)$$

*with the following properties:*

*(1) $w_1 = \cdots = w_m = \lambda$;*

*(2) Every $R_i$, with the exception of the one associated to the skin membrane, contains precisely one antiport rule (which can be in fact a symport rule); the $R_i$'s corresponding to elementary membranes contain only one symport rule of type $(x, in)$.*

*Conversely, every P system as above is described by a string in $SMDC(V)$.*

*Proof.* The direct implication follows from the definition of strings in $MDC(V)$, and the way the associated tree is constructed. All internal nodes have edges with double labels (none is $(\lambda, \lambda)$), and from here follows the existence of precisely one antiport rule. The root node coresponds to the "skin" parentheses, and thus has no rules.

For the converse, take a P system, not necessarilly fully specified

$$P = (V, \mu, w_1, \cdots, w_m, E, R_1, \cdots, R_m, i_0),$$

which satisfies (1) and (2). Consider a tree structure which describes the membrane structure $\mu$ and label its nodes with integers from 1 to $m$ corresponding to the depth-first traversal. The root will have label 1. For every edge which enters a node $i$, with $i \geq 2$, take the unique rule $R_i = \{(a_i, in; b_i, out)\}$ and put the label $(a_i, b_i)$ on the edge, $a_i$ on its left side, and $b_i$ on its right side. Now, make a depth-first traversal of the obtained tree, writing:

– $[_1$ when we start from the root, and $]_1$ when we end the traversal in the root;

– $[_i a_i$ every time we descend from a node to its son $i$, and $a_i$ is the left label of the edge on which we are descending;

– $b_i]_i$ every time we go up from a son $i$ to its parent, and $b_i$ is the right label of the edge which we are climbing.

The obtained string is in $SMDC(V)$ because, with the exception of the external brackets $[_1]_1$, which represent the skin membrane and for which $R_1 = \emptyset$, all the other nodes have precisely one rule associated to them. □

Note that there can be several trees, and thus several strings in $SMDC(V)$, which describe the same P system.

*Example 3.* Apply the above construction of a string, to the tree in Fig. 1. We obtain the string in Example 1, embedded in a supplementary pair of external brackets $[_1\ ]_1$, i.e., $[_1\ [_2\ a\ [_3\ a\ [_4\ ab\ ]_4\ [_5\ ab\ [_6\ c\ ]_6\ b\ ]_5\ c\ ]_3\ c\ ]_2\ [_7\ a\ ]_7\ ]_1$.

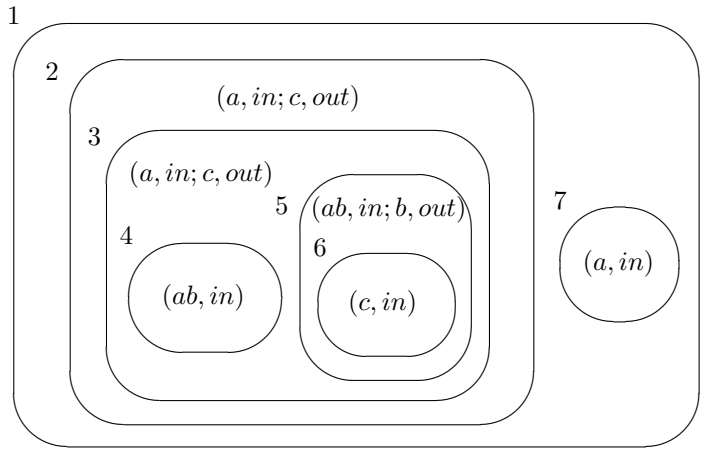



**Fig. 2**

*Example 4.* The membrane structure associated to the tree in Fig. 1, which was constructed from the string in Example 1, is depicted in Fig. 2.

From the previous lemma and the definitions of bracketed contextual grammars, we also have:

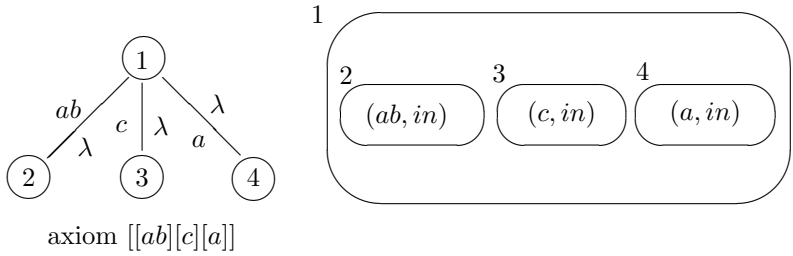
**Lemma 2.** *Let $G$ be a bracketed or a fully bracketed contextual grammar. Let $x \in A$ be an axiom of the form $x = [x']$ with $x' \in MDC(V)$. Then, to every $y \in (V \cup B)^*$ such that $x \Longrightarrow_G^* y$ there will correspond a unique P system*

$$P_y = (V, \mu, w_1, \cdots, w_m, E, R_1, \cdots, R_m, i_0)$$

*with the following properties:*

*(1) $w_1 = \cdots = w_m = \lambda$;*

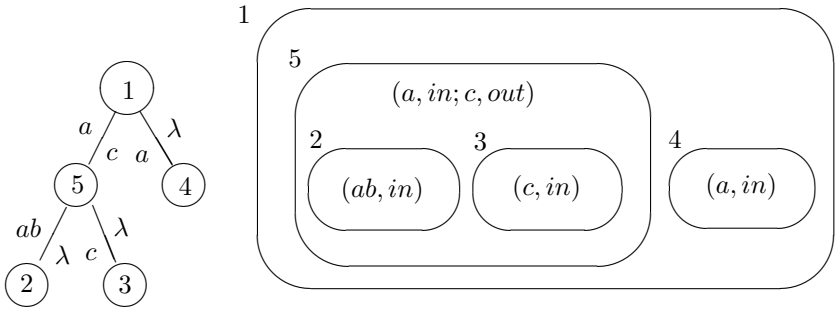
*(2) Every $R_i$, with the exception of the one corresponding to the skin membrane, contains precisely one antiport rule (which can be in fact a symport rule); the $R_i$'s corresponding to elementary membranes contain only one symport rule of type $(w, in)$.*




axiom $[[ab][c][a]]$

**Fig. 3**

The lemma follows easily from the fact that from a string in $MDC(V)$, by applying one derivation step in $G$, we obtain a string still in $MDC(V)$, and the fact that the axiom with which we start the derivation is in $SMDC(V)$ (and thus has a skin, and corresponds to a P system).



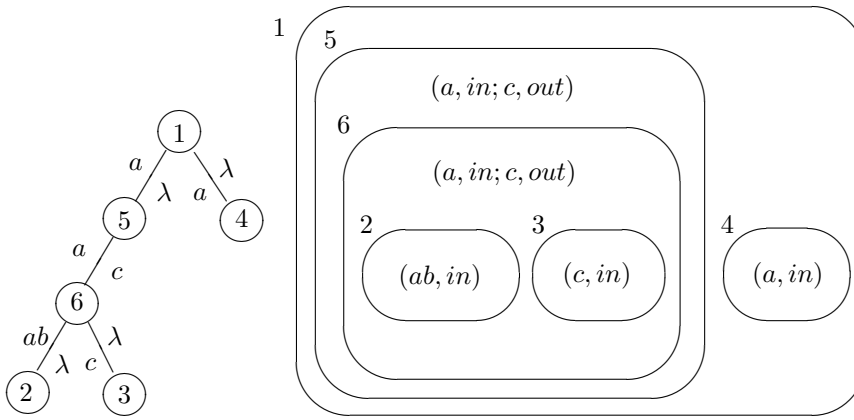

one application, $p_2$ new membrane 5

**Fig. 4**

The lemma implies that, to a sequence of derivations in a bracketed or fully bracketed contextual grammar $G$,

$$x \Longrightarrow_G y_1 \Longrightarrow_G y_2 \Longrightarrow_G \cdots \Longrightarrow_G y_n$$

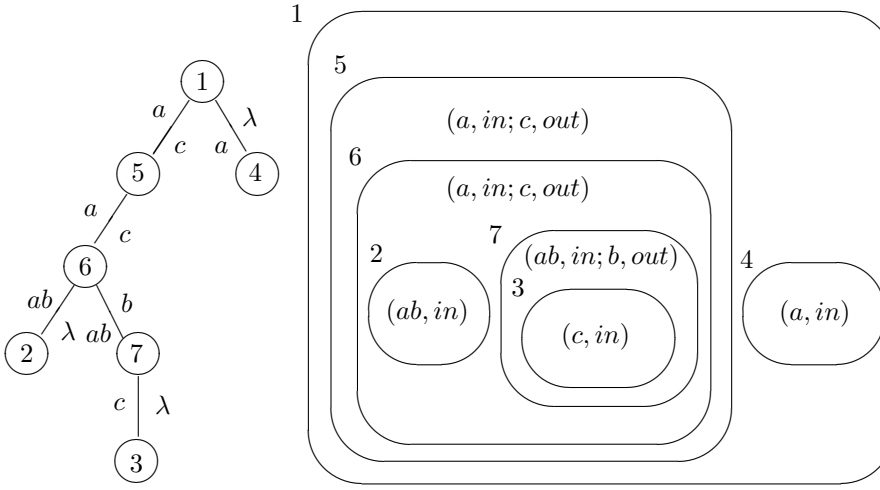
we can attach a sequence of P systems, $\{P_x, P_{y_1}, \cdots, P_{y_n}\}$, each one of the type described in the lemma.




second application, $p_2$ new membrane 6

**Fig. 5**

*Example 5.* The trees and the corresponding P systems associated to every string in the derivation in the bracketed contextual grammar from Example 2, are depicted in Figs. 3, 4, 5, and 6 respectively.




third application, $p_1$ new membrane 7

**Fig. 6**

Note that in the grammar of Example 2, after one application of the derivation rule $p_1$, rule $p_2$ is no longer applicable, so all the derivation sequences are of the type "apply $r$ times rule $p_2$, and then $s$ times rule $p_1$" ($r$ can be 0). This

means that all the associated P systems have the same "general shape", and we describe it in Fig. 7, the P system obtained from the original one after $r$ applications of rule $p_2$, followed by $s$ applications of rule $p_1$.

To summarize what we have obtained so far: bracketed and fully bracketed contextual grammars (by the way, one of the most actively researched branches of contextual grammars) generate special types of strings – strings which are descriptions of a special class of tree structures, with doubly labelled edges – trees which in turn are describing the membrane structure of special types of P systems.

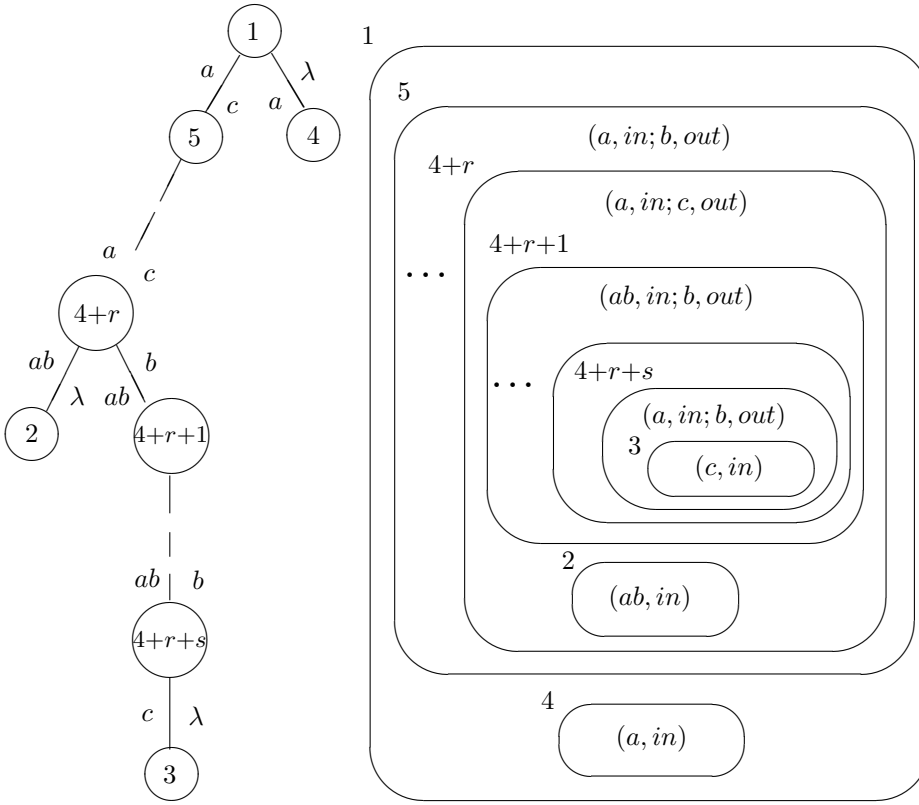



**Fig. 7**

The only problem is that the P systems generated so far are too poor to be able to perform internal computations, and thus are devoid of interest. We will remedy this problem in the next sections, by trying to find "good" string-descriptions for interesting types of P systems, and, by exploiting (and even enriching) the typology of contextual grammars, to find "good" generative devices for families of P systems.

### 3.3 New Types of Bracketed Strings

In this section we consider other types of strings, richer in structure than the (minimally) Dyck covered ones, strings which are able to describe P systems with a richer structure.

Let $B_s = \{[, |, ]\}$ be the alphabet containing, beside a pair of brackets, a special symbol, $|$, called separator.

**Definition 7.** *We call* Dyck language with one separator *the language* $D_B^s \subseteq B_s^*$ *defined by the following recursive relations:*

*(i)* $[\,|\,] \in D_B^s$;

*(ii) if* $x_1, x_2, \cdots, x_n \in D_B^s$, *then* $[x_1 x_2 \cdots x_n] \in D_B^s$ *for any* $n \geq 1$.

*Example 6.* The string

$$[\,[\,[\,|\,]\,[\,|\,]\,]\,[\,[\,|\,]\,]\,]$$

is in $D_B^s$. Actually, we can associate natural numbers starting from 1, to every pair of parentheses $[, ]$ and in the case of parentheses with separator, we associate the same number to the separator. For the above string we obtain:

$$[_1\,[_2\,[_3\,|_3\,]_3\,[_4\,|_4\,]_4\,]_2\,[_5\,[_6\,|_6\,]_6\,]_5\,]_1.$$

Let $V$ be another alphabet, disjoint from $B_s = \{[, |, ]\}$.

**Definition 8.** *A string* $x \in (V \cup B_s)^*$ *is called a* well-covered string over $V$, with respect to $D_B^s$ *iff the following hold:*

*(a)* $x = [y]$, *with* $y \in (V \cup B_s)^*$;

*(b)* $pr_{B_s}(x) \in D_B^s$.

Note that Dyck covered strings are in particular well-covered strings over $V$ w.r.t. the Dyck language $D_B$.

Denote by $DS(V)$ the set of all well-covered strings over $V$ w.r.t. $D_B^s$. Denote by $SDS(V)$ the set of strings in $DS(V)$, with skin, i.e., $x = [y]$ with $y \in DS(V)$.

*Example 7.* The string

$$[_1\,a[_2\,b[_3\,c|_3\,d]_3\,x[_4\,e|_4\,f]_4\,g]_2\,y[_5\,h[_6\,k|_6\,l]_6\,m]_5\,n]_1$$

is in $DS(V)$. Its projection over $D_B^s$ is precisely the string in Example 6.

To every string $w \in DS(V)$ we associate a tree $\tau(x)$ with doubly labelled edges, labels from $V^*$, and with nodes containing, on one hand integer labels, which identify each node unambiguously, and on the other hand, labels from $V^*$. We construct the tree $\tau(x)$ from $x$ according to the following rules:

– draw the root, and label it with integer 0;

– scan $x$ from left to right attaching an integer label (the next integer) to every $[$ encountered, and grow $\tau(x)$ with the rules:

(down) for each maximal $[w$, with $w \in V^*$ (which means that after $w$ we have a letter in $B_s$), draw a new edge, starting in the current node of the partially

constructed $\tau(x)$, mark the edge with $w$ on its left, and place it to the right of the node where it started; the new node, at the end of the edge, will have the integer label associated to the encountered [ ;

(up) for each maximal $w]$, or $|w]$, with $w \in V^*$, climb the current edge and write label $w$ on its right;

(stay) for each maximal $]w[$, with $w \in V^*$, stay in the current node and put label $w$ in it (we are in a node, since we just climbed to one, and will descend into another son of it).

For every such tree, we have a corresponding membrane system with one symport/antiport rule per membrane, and objects from $V$, constructed with the rules:

- the root node, labelled 0, can be considered as an *outer skin* and will play the role of the environment $E$ (we will not draw it);

- the root node 0 has only one son, with label 1, and the tree with root 1 is the usual tree description of the membrane structure $\mu$ of a P system;

- there is a membrane corresponding to every internal node; since there is an edge which enters every node, and it is doubly labelled (let the pair $(x, y)$ compactly describe the label), we will associate to the membrane a unique antiport rule of the form $(x, in; y, out)$. (It can degenerate into a symport rule.) The rule associated to membrane 1 (the skin) will be the rule that governs the communication with the environment $E$.

The tree associated to the string in Example 7 and the corresponding P system are depicted in Fig. 8.

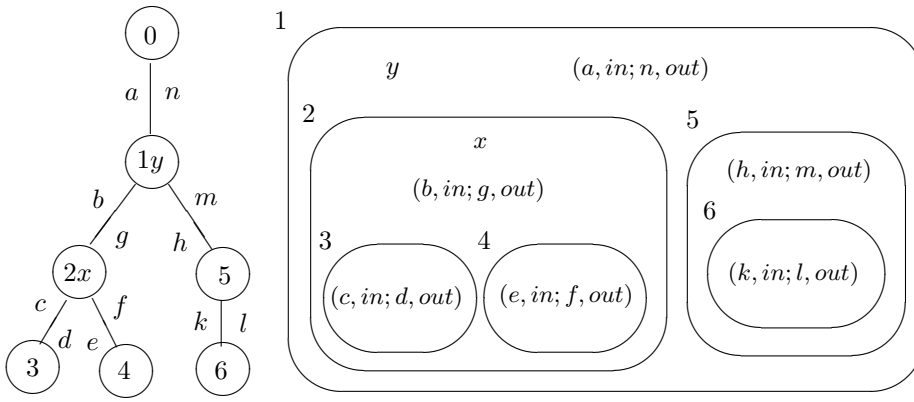



**Fig. 8**

**Lemma 3.** *To every string $x \in SDS(V)$ there corresponds a P system*

$$P_x = (V, \mu, w_1, \cdots, w_m, E, R_1, \cdots, R_m, i_0)$$

*with the following properties:*

*(1) $w_i = \lambda$ for all $i$ which correspond to membranes which are either elementary, or are non-elementary, but have only one sub-membrane;*

*(2) Every $R_i$ contains precisely one antiport rule (which can be in fact a symport rule).*

*Conversely, every P system as above is described by a string in $SDS(V)$.*

*Proof.* Similar to the proof of Lemma 1. □

Note that if an internal node $i$ has $k_i$ sons, then it will have $k_i - 1$ strings in $V^*$ as labels, but their interpretation as objects in the membrane $i$ is independent on the number of strings.

Note also that the tree associated to a string is unique, but it is *not the unique* description of a P system. Several trees, and thus several strings in $SDS(V)$ will describe the same P system.

We intend to use strings in $SDS(V)$ as axioms for (new types of) bracketed contextual grammars. What we have accomplished with the price of one extra separator | is the possibility on having antiport rules of the most general type associated to elementary membranes, and, by relaxing the conditions which were imposed on minimally Dyck covered strings, we have obtained also objects inside some of the membranes. Still, as we will see, this is not powerful enough.

### 3.4 New Types of Contextual Productions

We consider in the sequel several types of contextual productions, i.e., pairs of selectors and contexts, and the derivations associated with them, and we study what we obtain in terms of P systems. (Obviously, the formalism will have to be enriched.) To keep the picture clear, we will consider (as much as possible) singleton selectors, and singleton contexts, and we do not bracket singletons.

**1. Contexts which generate new membranes**

This has been the case so far. Take the production $p = (y_1 \cdots y_n, ([u, v]))$, with $y_j \in SDS(V)$, for $1 \leq j \leq n$ and $u, v \in V^*$. Then we have:

$x \Longrightarrow_p y$ iff $x = x_1 x_2 x_3$ , $y = x_1[ux_2v]x_3$,

$x_2 = y_1 \cdots y_n$ and $y_j \in SDS(V)$, for $1 \leq j \leq n$.

One internal derivation step as above has the effect of embedding the membranes corresponding to $y_1, \cdots, y_n$ in a new membrane, with one antiport rule, $(u, in; v, out)$.

**2. Contexts which generate new elementary membranes**

Take the production $p = (\ ], (\lambda, [u|v])$. We have

$x\ ]\ y \Longrightarrow_p x\ ]\ [u|v]\ y$.

If $y$ is different from $\lambda$, then a new elementary membrane will be added at the same level as its selector ]. The new membrane has one antiport rule $(u, in; v, out)$.

**3. Contexts which add new rules to elementary membranes**

We will have to consider ; a new separator, in order to distinguish between rules.

Take as selector $S = \{\ |\ \}$ and the set of contexts $C_1 = \{(; u, v;)\}$, $C_2 = \{(; x, \lambda;)\}$, $C_3 = \{(; \lambda, y;)\}$. Denote by $p_i = (S, C_i)$. Since the selector | is to

be found only in elementary membranes, we can see how a derivation works by applying it to $[a|b]$:

$[a|b] \Longrightarrow_{p_1} [a; u|v; b]$
$\Longrightarrow_{p_2} [a; u; x|\lambda; v; b]$
$\Longrightarrow_{p_3} [a; u; x; \lambda|y; \lambda; v; b]$.

The application of $p_1, p_2, p_3$ has enriched the membrane with the antiport rule $(u, in; v, out)$, the symport rule $(x, in)$, and the symport rule $(y, out)$, respectively.

Note that applying the same derivation rule to the same selector several times does not enrich the corresponding membrane structure (since the rule was already added). Strings obtained with this kind of productions are *not* in $SDS(V)$.

**4. Adding objects to elementary membranes**

Use again the selector $|$, and as context some $C_w = \{(\lambda, w|)\}$ with $w \in V^*$. Then we have the derivation $[a|b] \Longrightarrow_{p_w} [a|w|b]$, and we interpret the result as the old elementary membrane, with antiport rule $(a, in; b, out)$, enriched with the objects $w$. Note that the produced string is no longer in $SDS(V)$, because it contains more than one appearance of the separator $|$ inside a pair of brackets.

For a finite set of contexts $C = \{(\lambda, w_1|), \cdots (\lambda, w_n|)\}$ we will have

$[a|b] \Longrightarrow [a|w_1| \cdots |w_n|b]$,

with the meaning: the elementary membrane has been enriched with the strings $\{w_1, \cdots, w_n\}$. Again, the derived string is not in $SDS(V)$.

**5. Strings as selectors**

Consider now a string $x \in V^*$ as selector for a context $(u, v)$ with $u, v \in V^*$. Depending on the place in which $x$ occurs, such a production can (1) either modify a rule, or (2) change the contents of a membrane. For instance:

(1) $[axb|y] \Longrightarrow [auxvb|y]$, or
(1') $y_1[axb[y_2 \Longrightarrow y_1[auxvb[y_2$.

In (1) the rule $(axb, in; y, out)$ has been changed into rule $(auxvb, in; y, out)$. Similarly, in (1') the left-hand-side of an antiport rule has been changed.

Or, we can have:

(2) $[a|rxs|b] \Longrightarrow [a|ruxvs|b]$,
(2') $a\ ]\ rxs\ [\ b \Longrightarrow a\ ]\ ruxvs\ [\ b$.

The rule in (2) has enriched an elementary membrane with objects $u$ and $v$, while in (2') a membrane which is not elementary has been enriched. (If we want to add objects where we have none, we will have to use different productions, and possibly different derivations.)

### 3.5 Enriched Bracketed Contextual Grammars

The discussion in the previous two subsections has shown that, if we want to represent P systems with a more complicated structure (several symport/antiport rules per membrane, strings inside membranes, and the possibility of adding rules and strings to given membranes), then we will need more complicated bracketed

strings than the well-formed ones over the Dyck language with one separator. We will need at least two separators, | as before, for the contents of a membrane, and ; a separator between rules, and we have to allow for several appearances of each one of them.

We will define now the notion of a *membrane expression* or *m-expression* for short.

Let $S = \{ [ , ] , | , ; \}$ be the *alphabet of separators* and $V$ (disjoint from $S$) the alphabet of objects.

**Definition 9.** *A string $x \in (V \cup S)^*$ will be called* an elementary m-expression over $V$ *iff $x$ is of the form*

$$x = [a_1; \cdots ; a_n | w_1 | \cdots | w_m | b_n; \cdots ; b_1],$$

*with $a_i, b_i \in V^*$, for $1 \leq i \leq n$, $w_j \in V^*$, for $1 \leq j \leq m$, and natural numbers $n, m \geq 0$.*

Such a string describes an elementary membrane, with $n$ symport/antiport rules $R = \{(a_i, in; b_i, out) \mid 1 \leq i \leq n\}$, and containing symbol-objects in the form of strings $\{w_j \mid 1 \leq j \leq m\}$. If $m = 0$, then the membrane has no objects in it, only rules, and we use only one separator |, i.e., $x$ becomes

$$x = [a_1; \cdots ; a_n | b_n; \cdots ; b_1].$$

If $n = 0$, then the membrane will have no rules, only strings, i.e., $x$ becomes

$$x = [\ | w_1 | \cdots | w_m |\ ].$$

An empty elementary membrane, with no strings and no rules, will be the m-expression [ | ], or [ ].

**Definition 10.** *A string $x \in (V \cup S)^*$ will be called* an m-expression over $V$ *iff*

*(1) either $x$ is an elementary m-expression;*

*(2) or there exist the m-expressions $\{e_k \mid 1 \leq k \leq t\}$, the strings $\{w_j \mid 1 \leq j \leq m\} \subseteq V^*$, and the pairs of strings $\{(a_i, b_i) \mid 1 \leq i \leq n\} \subseteq V^* \times V^*$ such that $x$ is of the form*

$$x = [a_1; \cdots ; a_n | w_1 | \cdots | w_m | e_1 | \cdots | e_t | b_n; \cdots ; b_1].$$

An m-expression describes a membrane structure with objects, symport/antiport rules for objects, and sub-membranes described by the m-expressions $\{e_k \mid 1 \leq k \leq t\}$.

The set of all m-expressions over $V$ will be denoted by $ME(V)$, and can be inductively constructed from the definition above.

*Remark 2.* Since there are several m-expressions which describe the same P system, we will actually work with *equivalence classes* of strings in $ME(V)$.

**Definition 11.** *On $ME(V)$ define the relation $\sim$ as the smallest equivalence relation for which, if $x$ is an m-expression*

$$x = [a_1; \cdots; a_n | w_1 | \cdots | w_m | e_1 | \cdots | e_t | b_n; \cdots; b_1],$$

*as in Definition 10, then the following hold:*

*(1)* $x \sim y$, *if* $y = [a_1; \cdots; a_n | w_{\sigma(1)} | \cdots | w_{\sigma(m)} | e_{\tau(1)} | \cdots | e_{\tau(t)} | b_n; \cdots; b_1]$, *where* $\sigma \in Perm(m)$ *and* $\tau \in Perm(t)$ *are permutations of* $m$, *respectively* $t$ *elements.*

*(2)* $x \sim z$, *if* $z = [\rho_1(a_1); \cdots; \rho_n(a_n) | w_1 | \cdots | w_m | e_1 | \cdots | e_t | \rho_{n+1}(b_n); \cdots; \rho_{2n}(b_1)]$, *where by* $\rho_j(x)$ *we have denoted a permutation of the letters of* $x$, *for* $1 \leq j \leq 2n$.

*(3)* $x \sim x'$ *if* $x' = [a_1; \cdots; a_n | w_1 \cdots w_m | e_1 | \cdots | e_t | b_n; \cdots; b_1]$ *(we have catenated the strings* $w_1, \cdots, w_m$*.)*

*(4)* $x \sim x''$ *if* $x'' = [a_1; \cdots; a_n | \rho(w_1 \cdots w_m) | e_1 | \cdots | e_t | b_n; \cdots; b_1]$, *where* $\rho(w)$ *denotes a permutation of the letters of* $w$.

*(5)* $x \sim x'''$, *if* $x'''$ *is obtained by a combination of* (1), (2), (3) *and/or* (4) *above, or by permuting between them strings* $w_j$ *with m-expressions* $e_k$.

We will denote by $\widehat{x}$ the class of the string $x \in ME(V)$.

An m-expression as in Definition 10, where each $w_i$ is a word over a one-letter alphabet, will be called an m-expression *in canonical form.*

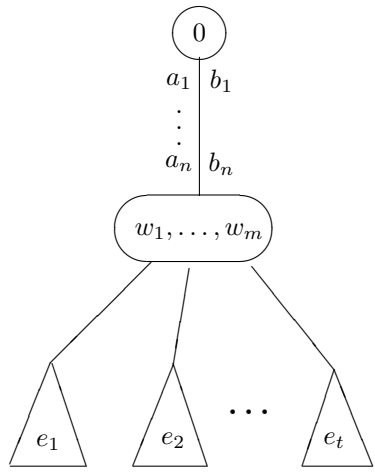



**Fig. 9**

*Remark 3.* To every element $\widehat{x}$ of $ME(V)/\sim$ there corresponds an unordered tree, $\tau(x)$, in the following recursive way:

(1) if $x$ is an elementary m-expression, then $\tau(x)$ has one root node labelled with integer 0, and only one descendant, a node labelled with integer 1, and one edge entering into it; the edge has $n$ pairs of labels $\{(a_i, b_i) \mid 1 \leq i \leq n\}$, and node 1 has internal labels $\{w_j \mid 1 \leq j \leq m\} \in V^*$. Note that node 1 is a leaf.

(2) if $x$ is a non-elementary m-expression, then it has nodes 0 and 1 as above, and node 1 has $t$ descendants (trees) represented by the m-expressions $e_1, \cdots, e_t$. The leaves of such a tree will correspond to elementary m-expressions.

Figure 9 represents a tree, corresponding to the m-expression in Definition 10.

*Remark 4.* The set of symport/antiport rules associated to a membrane can be either an *unordered set*, or an *ordered set*; in this latter case, there is a natural *total order* on rules. We make the convention that the most "external" ones are stronger than the "internal" ones, i.e.,

$$(a_1, in; b_1, out) > (a_2, in; b_2, out) > \cdots > (a_n, in; b_n, out).$$

Our definition above of the equivalence relation $\sim$ does not allow commutation of rules.

**Lemma 4.** *To every $\widehat{x} \in ME(V)/\sim$ there corresponds a P system with symbol-objects in $V$, and with a totally ordered set of symport/antiport rules associated to every membrane.*

*Reciprocally, to every P system as above we can attach an element $\widehat{x} \in ME(V)/\sim$.*

**Contextual derivations in $ME(V)/\sim$**

A derivation in a contextual grammar produces from a *string*, $x$, another *string*, $y$, using a certain production $p$ (or alternatively, a context selected by the selection function). Both strings are over $V$. Even in the case of bracketed contextual grammars, there can be emphasis on the *string language* generated by the grammar (which consists only of the string projections over $V$), or on the *bracketed language*, which consists of pairs of strings over $V$ *and* the associated tree.

We are interested here mainly in the *tree part* of words in the bracketed language; moreover, we are interested in *equivalence classes* of such trees, an equivalence class containing all possible string-descriptions of a P system.

Because of this, even if we will consider contextual grammars which describe and preserve (under derivation) m-expressions, we will look for the results of derivations on *classes* of m-expressions.

**Definition 12.** *Let $G$ be a contextual grammar of an arbitrary type (internal, external, total, n-contextual, in the functional or modular presentation, insertion grammar, etc.) over the alphabet $V \cup S$, where $V$ is an arbitrary alphabet, and $S$ is the alphabet of separators defined above. Denote by $A$ the set of axioms of $G$, and by $\Longrightarrow$ the derivation relation associated with $G$. We say that $G$ is an* enriched bracketed contextual grammar *of the corresponding type (internal, external, total, n-contextual, insertion, etc) iff $A \subseteq ME(V)$, and for any $x \in ME(V)$ and for any derivation $x \Longrightarrow y$, we have $y \in ME(V)$.*

**Definition 13.** *If $G$ is an enriched bracketed contextual grammar and $\Longrightarrow$ is the derivation relation associated with $G$, we call* enriched contextual derivation *induced by $G$ the following binary relation on $ME(V)/\sim$:*

$$\widehat{x} \Longrightarrow_{eb} \widehat{y} \text{ iff } \widehat{x} \in ME(V)/\sim, \widehat{y} \in ME(V)/\sim \text{ and}$$
$$\text{there exist representatives } x' \in \widehat{x} \text{ and } y' \in \widehat{y}$$
$$\text{such that } x' \Longrightarrow y'.$$

**Enriched contextual productions and derivations**

We present in the sequel several types of contextual productions and we analyze what we obtain from them in terms of P systems. Our goal is to construct contextual grammars, which have m-expressions as axioms, and such that the derivation associated to production rules (or selection functions) leads to other m-expressions. In the rest of this section, we will consider that $G$ is a contextual grammar of an appropriate type.

**1. Productions which add non-elementary membranes**

Let $x$ be an m-expression in its canonical form

$$x = [a_1; \cdots; a_n|w_1| \cdots |w_m|e_1| \cdots |e_t|b_n; \cdots; b_1].$$

Consider the following contextual production in $G$:

$$p = (e_{i_1}| \cdots |e_{i_k},\ ([u|, |v])),$$

with $u, v \in V^*$ and $e_{i_j} \in ME(V)$ for all $1 \leq j \leq k$ (we may represent the sets of selectors and respectively contexts from a contextual production by one element each, as long as they are both singletons).

If $\{e_{i_1}, \cdots, e_{i_k}\} \subseteq \{e_1, \cdots, e_t\}$, then there exist m-expressions

$$\{f_1, \cdots, f_{t-i_k}\} = \{e_1, \cdots, e_t\} \backslash \{e_{i_1}, \cdots, e_{i_k}\}$$

and $x' \in \widehat{x}$ such that

$$x' = [a_1; \cdots; a_n|w_1| \cdots |w_m|e_{i_1}| \cdots |e_{i_k}|f_1| \cdots |f_{t-i_k}|b_n; \cdots; b_1].$$

To $x'$ we can apply the internal string derivation $\Longrightarrow$ of $G$, and obtain $x' \Longrightarrow y$, with

$$y = [a_1; \cdots; a_n|w_1| \cdots |w_m|[u|e_{i_1}| \cdots |e_{i_k}|v]|f_1| \cdots |f_{t-i_k}|b_n; \cdots; b_1].$$

We have thus the enriched internal derivation $\widehat{x} \Longrightarrow_{eb} \widehat{y}$ which has the effect of embedding all membranes $\{e_{i_1}, \cdots, e_{i_k}\}$ which are sub-membranes of $\widehat{x}$, into a new membrane with one antiport rule $(u, in; v, out)$.

Adding a new membrane with several rules is also possible, using productions of the form:

$$p = (e_{i_1}| \cdots |e_{i_k}\ ,\ ([u_1; \cdots; u_s|, |v_s; \cdots; v_1])).$$

The new membrane will have a (totally ordered) set of antiport rules

$$\{(u_1, in; v_1, out) \geq \cdots \geq (u_s, in; v_s, out)\}.$$

Instead of the modular presentation we could have used the functional presentation and define the selection function $\phi$ by:

$$\phi(e_{i_1}| \cdots |e_{i_k}) \ = \ ([u_1; \cdots; u_s|, |v_s; \cdots; v_1]),$$

where in the right hand-side we have a singleton, for which we omitted, as usual, the brackets.

**2. Productions which add elementary membranes**
Consider the following contextual production in $G$:

$$p = (\ |\ , (\lambda, [u_1; \cdots; u_s | w | v_s; \cdots; v_1]\ |\ )\ ).$$

If

$$x = [a_1; \cdots; a_n | w_1 | \cdots | w_m | e_1 | \cdots | e_t | b_n; \cdots; b_1],$$

then we obtain an enriched internal derivation $\widehat{x} \Longrightarrow_{eb} \widehat{y}$ where:

$$y = [a_1; \cdots; a_n | w_1 | \cdots | w_m | e_1 | \cdots | e_t | [u_1; \cdots; u_s | w | v_s; \cdots; v_1]\ |\ b_n; \cdots; b_1].$$

Note that in the internal string-derivation the membrane can be put anywhere between strings $w_j$ or sub-membranes $e_k$ (in other words, the production can use *any* of the selectors | present in the expression), but we can find a representative $y$ as above in canonical form.

The selection can be enriched with further conditions. Adding a new elementary membrane can be conditioned on certain factors, and this can be modelled by appropriately changing the selector of the production (or the domain of the selection function $\phi$). For instance, if we want to insert an elementary membrane inside another membrane, only if the latter contains some specified string object $x \in V^*$, then we can use the production:

$$p = (\ |x|\ , (\lambda, [u_1; \cdots; u_s | w | v_s; \cdots; v_1]\ |\ )\ ),$$

or the selection function

$$\phi(\ |x|\ ) = (\lambda, [u_1; \cdots; u_s | w | v_s; \cdots; v_1]\ |\ ).$$

All the above derivations are internal ones. More complicated conditions can be formulated and modelled with total contextual selection functions and derivations, or with $n$-contextual ones, or even with insertion grammars.

**3. Productions which add new rules to membranes**
The new rules (of symport/antiport type only) will take the form of contexts

$$C = \{(; u_1, v_1;), \cdots, (; u_s, v_s;)\}.$$

The domain of the selection function $\phi$ "decides" where the new rules are to be added. But we should be careful with the selection function. For instance, $\phi(\ |\ ) = C$, which was considered in Section 3.4, can be used to associate rules to elementary membranes with no string objects in them. But if we allow this choice, the derivation could be applied in "wrong places", getting us out of the set $ME(V)$.

If the whole content (strings and sub-membranes) of a membrane plays a role in the choice of new rules to be added, then we can use a selection function

$$\phi(|w_1| \cdots |w_m| e_1 | \cdots | e_t|) = C.$$

For

$$x = [a_1; \cdots; a_n | w_1 | \cdots | w_m | e_1 | \cdots | e_t | b_n; \cdots; b_1],$$

the sequence of internal string-derivations

$$x \Longrightarrow_1 y_1 \Longrightarrow_2 y_2 \cdots \Longrightarrow_s y_s$$

(where $\Longrightarrow_j$ denotes the application of the context $(; u_j, v_j;)$) will yield:

$$\begin{aligned} y_1 &= [a_1; \cdots; a_n; u_1 | w_1 | \cdots | w_m | e_1 | \cdots | e_t | v_1; b_n; \cdots; b_1], \\ &\vdots \\ y_s &= [a_1; \cdots; a_n; u_1; \cdots; u_s | w_1 | \cdots | w_m | e_1 | \cdots | e_t | v_s; \cdots; v_1; b_n; \cdots; b_1]. \end{aligned}$$

Recall that the set of rules is totally ordered, and thus the order in which the contexts are applied does count.

If the presence of other rules in the membrane determines the choice or the rules to be added, or if only *part* of the content of a membrane determines the choice, the formulation of the selection function could be either more complicated, or even impossible in the frame of *internal* derivations. But again, we could use total or $n$-contextual grammars.

**4. Productions which add new objects to membranes**

Adding objects in the form of strings $w \in V^*$ to a membrane, without further constraints, can be accomplished with the selection function

$$\phi(\ |\ ) = (\lambda, w|)\ ,\ \text{with}\ \ w \in V^*.$$

An internal derivation which uses it will enrich the string content of a membrane, a fact which can be easily shown.

If we want to condition the addition of new objects on the previous existence of other objects, like $x \in V^*$, and/or sub-membrane structures, like $e \in ME(V)$, then we will use the respective selection functions:

$$\begin{aligned} \phi(\ |\ x\ |\ ) &= (\lambda, w|)\ ,\ \text{with}\ \ w \in V^*, \\ \phi(\ |\ e\ |\ ) &= (\lambda, w|)\ ,\ \text{with}\ \ w \in V^*. \end{aligned}$$

Again, conditioning the addition of objects on the existence of certain rules will have to be formulated in a total contextual or $n$-contextual setting. For instance, conditioning the addition of objects $w$ on the whole set of rules of a membrane could be modelled by the following total selection function:

$$\phi([\ a_1; \cdots; a_n\ ,\ |\ z\ |\ ,\ b_n; \cdots; b_1]) = (\lambda, w|)\ ,\ \text{with}\ \ w \in V^*.$$

If

$$x = [\ a_1; \cdots; a_n\ |\ z\ |\ b_n; \cdots; b_1\ ],$$

then the total string-derivation $x \Longrightarrow y$ will produce

$$y = [\ a_1; \cdots; a_n\ |\ z\ |\ w\ |\ b_n; \cdots; b_1\ ].$$

### 3.6 Remarks on the Generative Approach

In the previous section we have formally defined the set of strings $ME(V)$ and the equivalence relation $\sim$, which help us express, using elements of $ME(V)/\sim$, all P systems with symbol objects and totally ordered sets of symport/antiport rules associated to membranes. We have worked with totally ordered sets of rules only because we will need such sets of rules in the second part of the paper, more precisely in subsections 5.3, 5.4, and 5.5.

In order to work with P systems with symport/antiport rules *without priority relations*, the formalism needs very little changes.

Let $ME(V)$ be as above, and $S$ the same set of separators.

On $ME(V)$ define the relation $\sim_1$ as the smallest equivalence relation which contains $\sim$, and for which, if $x$ is as an m-expression

$$x = [a_1; \cdots; a_n | w_1 | \cdots | w_m | e_1 | \cdots | e_t | b_n; \cdots; b_1],$$

as in Definition 10, then the following holds:

(6) $x \sim_1 y$, for $y = [a_{\sigma(1)}; \cdots; a_{\sigma(n)} | w_1 | \cdots | w_m | e_1 | \cdots | e_t | b_{\sigma(n)}; \cdots; b_{\sigma(1)}]$, where $\sigma \in Perm(n)$ is a permutation of $n$ elements.

**Lemma 5.** *To every $\widehat{x} \in ME(V)/\sim_1$ there corresponds a P system with symbol objects in $V$, and with symport/antiport rules (without priority relations) associated to every membrane.*

*Reciprocally, to every P system as above, we can attach an element $\widehat{x} \in ME(V)/\sim_1$.*

The definitions of enriched bracketed contextual grammars of the previous section remain unchanged. If $G$ is such a grammar, and $\Longrightarrow$ is its derivation relation, then the following binary relation on $ME(V)/\sim_1$ can be considered:

$$\widehat{x} \Longrightarrow_{eb} \widehat{y} \text{ iff } \widehat{x} \in ME(V)/\sim_1, \widehat{y} \in ME(V)/\sim_1 \text{ and there exist representatives } x' \in \widehat{x} \text{ and } y' \in \widehat{y} \text{ such that } x' \Longrightarrow y'.$$

This is nothing else but an *enriched contextual derivation* working on $ME(V)/\sim_1$ (instead of $ME(V)/\sim$), and thus able to generate families of P systems with symport/antiport of the standard type considered in the literature.

A type of question which can now be meningfully asked is the following. Suppose we have a given "family" of P systems with communication, $\{\Pi_n \mid n \geq 1\}$, specified in a certain manner. Can we find:

(i) the "simplest" P system $A$?

(ii) the "simplest" enriched bracketed contextual grammar $G$, with $\{A\}$ its axiom set?

(iii) the shortest derivations in $G$, such that $A \Longrightarrow^* \Pi_n$ for every $n$?

or maybe even such that:

(iv) $A \Longrightarrow \Pi_{\sigma(1)} \Longrightarrow \Pi_{\sigma(2)} \Longrightarrow \cdots \Longrightarrow \Pi_{\sigma(n)}$, for a permutation $\sigma \in Perm(n)$?

Let $G$ be an enriched bracketed contextual grammar, with $V$ the alphabet of objects, $S$ its alphabet of separators, denote by $Ax(G)$ its set of axioms, and let $\Longrightarrow$ denote its derivation relation. We can consider several types of "families" of P systems generated by $G$:

(1) Sequences of P systems obtained by succesive derivations from an axiom:

$$\Pi_1 \Longrightarrow \Pi_2 \Longrightarrow \cdots \Longrightarrow \Pi_n \Longrightarrow \cdots$$

(2) The "language" generated by $G$:

$$L(G) = \{\Pi \mid \Pi_1 \Longrightarrow^* \Pi, \text{ for some } \Pi_1 \in Ax(G)\}.$$

(3) All P systems obtained by derivations in $G$, of a given length $m$, (i.e., all "words" of length $m$ of the above language):

$$L_m(G) = \{\Pi \mid \Pi_1 \Longrightarrow^m \Pi, \text{ for some } \Pi_1 \in Ax(G)\}.$$

Having such families of P systems, we can now meaningfully address questions such as finding similarity/resemblance relations between P systems.

Also, having a "compact" description in the form of a grammar, we can attack problems of descriptional complexity.

Investigating the computational power of such families is also a topic for further research.

The approach we have proposed illustrates a very general idea: that, once we have a "good" string-description of a certain type of P systems, and a contextual grammar mechanism working on the appropriate types of strings (with derivations which keep us inside the same class of strings), the contextual grammar can be used to generate families of P systems belonging to the same class.

As illustrated in [2], we need *not* restrict ourselves to P systems with communication, and also *not* to enriched bracketed contextual grammars.

## 4 Dynamic P Systems

We move in this section, from the purely generative approach adopted so far, towards considering also computational aspects of the families of P systems we generate.

The contextual grammar mechanisms previously considered, can be thought of (and used as) *descriptive* tools for evolutionary processes, intrinsic to the system itself, and/or triggered by some environmental changes. A P system which performs computations inside it, is also "changing" (by passing from one configuration to the next one), but the changes are not so dramatic if they do not involve creation of new membranes, massive addition of new rules and new objects. These types of "dramatic" changes can be described by the types of contextual grammars we have proposed. A pattern of alternating dramatic and non-dramatic changes emerges as the most general description of an "evolutionary" process. The concepts of dynamic P system and dynamic computation sequence, strongly linked together, try to capture this spirit.

We have constructed the set $ME(V)/\sim$ of equivalence classes of strings describing P systems with (ordered) sets of symport/antiport rules, and with objects inside. The enriched bracketed contextual grammars introduced have the properties that:

(1) their axioms are contained in $ME(V)/\sim$;

(2) their associated derivation relation keeps us inside $ME(V)/\sim$.

Then, if $G$ is such an enriched bracketed contextual grammar, repeated applications of derivations in $G$ generates a sequence of P systems of the same type, which have evolved one from the other in a coherent manner.

Note that this can be formulated in a much more general setting.

Suppose we have a set of well-formed strings (well-formed according to some formal definition), denoted $Exp(V)$, over an alphabet $V$, and containing also separators. Among the separators we will use the brackets $\{[,]\}$, to describe the membrane structure of the system, but we can use also other kinds of separators (like we used | and ; to construct $ME(V)/\sim$). The strings in $Exp(V)$ describe some particular type of P system.

Suppose that we also have a contextual grammar $D$ of a certain type (internal contextual, total contextual, insertion grammar, etc.) with the following properties:

(1) its set of axioms is contained in $Exp(V)$;

(2) its derivation relation, $\Longrightarrow_D$, keeps us inside the the set of well-formed strings, i.e., if $x \in Exp(V)$ and $x \Longrightarrow_D y$, then $y \in Exp(V)$.

Then, starting from a P system described by an axiom of $D$, and applying repeteadly derivations of $D$, we obtain a sequence of P systems which have "evolved" in a coherent manner from the original one.

We want to let the P systems generated by an enriched bracketed contextual grammar, or by a more general grammar as $D$ considered above, make also internal computations.

We can consider several types of internal computations:

– we will call *one-step computation* what is called in the standard P systems literature a *transition*;

– we call *stable computation* a sequence of transitions which reaches a stable configuration;

– we call *halting computation* a sequence of transitions which reaches a halting configuration, a configuration in which no rule is applicable. (Recall that in the standard literature this is the same as succesful computation.)

**Definition 14.** *A* dynamic P system with halting (respectively stable, respectively one-step) computations, associated to the grammar $D$ *will be a sequence of P systems* $\{\Pi_n \mid n \geq 1\}$ *such that:*

*(i)* $\Pi_1$ *is an axiom of* $D$*;*

*(ii) for each* $i \geq 1$*, let* $\Pi_i \rightsquigarrow C\Pi_i$ *denote a sequence of internal computations in* $\Pi_i$ *of the halting (respectively stable, respectively one-step) type, i.e.,* $C\Pi_i$ *is a halting (respectively stable, respectively the next) configuration of* $\Pi_i$*;*

*(iii) for* $i \geq 2$*, each* $\Pi_i$ *is obtained from* $\Pi_{i-1}$ *by the derivation* $C\Pi_{i-1} \Longrightarrow_D \Pi_i$ *in the grammar* $D$*.*

*The* dynamic computation sequence *associated to the above system is the following alternating sequence of derivations in $D$ and halting (respectively stable, respectively one-step) computations:*

$$\Pi_1 \rightsquigarrow C\Pi_1 \Longrightarrow_D \Pi_2 \rightsquigarrow C\Pi_2 \Longrightarrow_D \ldots \Longrightarrow_D \Pi_i \rightsquigarrow C\Pi_i \Longrightarrow_D \ldots .$$

Note that, in order to be able to construct such a dynamic computation sequence, not only the systems $\Pi_i$, but also $C\Pi_i$, must have string-descriptions in $Exp(V)$; the internal computations considered must be able to let the derivation mechanisms of the grammar $D$ work.

If internal computations in $\Pi_i$ are not possible, then we denote this by $\Pi_i = C\Pi_i$.

A dynamic P system is *finite* if the sequence of P systems is finite in the usual sense, i.e., $\{\Pi_n \mid 1 \leq n \leq m\}$. Its associated computation sequence will in this case also be finite.

A dynamic P system is *stationary* if the sequence $\{\Pi_n \mid n \geq 1\}$ is stationary in the usual sense, i.e., there exists an index $m$ such that, for all $k \geq m$, $\Pi_k = \Pi_{k+1}$. This means that neither internal computations, nor derivations in $D$ are possible in $\Pi_m$, respectively $C\Pi_m$.

Other versions of dynamic P systems can be obtained, by considering, instead of stable computations, computations which are stable w.r.t. a sub-alphabet of objects $W \subset V$, or w.r.t. a subset of rules.

*Remark 5.* The one-step computation and the halting computation remind of the modes $= k$, and $t$ respectively, from CD grammar systems area (see [4]). This also suggests the research topic of investigating dynamic P systems with other computation modes, inspired from the CD grammar systems area, such as $*$, $\leq k$, $\geq k$. Also, more complicated patterns for combining derivations and internal computations can be considered and studied.

We can use the concept of dynamic P system to solve, using P systems, *dynamic* problems. Suppose the set $X$ of input data for a problem is not available entirely at a certain moment, but is given step-wise. Then, we can imagine a mechanism, described by a derivation in a contextual grammar, which is responsible, among other things, of feeding a new data, $x \in X$, into a computing device based on P systems. The whole dynamic P system will compute: while the derivation steps are responsible for feeding new data, and creating conditions for their appropriate processing, the internal computation steps will "solve" the problem for the partial input set available.

If $X = \{x_1, \cdots, x_m, \cdots\}$ is the set of incoming input data, and we have a grammar whose derivations, denoted $\Longrightarrow_x$, are responsible for "feeding data $x$", then the above mechanism can be described as the following (abstract) dynamic computation sequence:

$$\begin{aligned} \Pi_1(x_1) \quad \rightsquigarrow \quad & C\Pi_1(x_1) \Longrightarrow_{x_2} \Pi_2(x_1, x_2) \rightsquigarrow C\Pi_2(x_1, x_2) \Longrightarrow_{x_3} \ldots \\ & \ldots \Longrightarrow_{x_i} \Pi_i(x_1, \cdots, x_i) \rightsquigarrow C\Pi_i(x_1, \cdots, x_i) \Longrightarrow_{xi+1} \ldots . \end{aligned}$$

If $X$ is finite, the sequence above will also be finite.

We will illustate this idea in the next section, with the (dynamic) problem of sorting. We will construct dynamic P systems associated to enriched bracketed contextual grammars, working on the set of well-formed membrane expressions $ME(V)$, systems whose associated computation sequence solves the problem of sorting for a dynamically given set of input data $X$.

A few words on the deterministic/non-deterministic aspects present in this new concept: in a dynamic P system, non-determinism can arise, on one hand, from the intrinsic non-determinism of P systems, during the internal computation step $\rightsquigarrow$, and, on the other hand, from the non-deterministic behavior of the derivations in the grammar $G$. In the applications which follow, we have avoided this second kind of non-determinism, by using deterministic contextual grammars, with singleton axiom sets.

## 5 Applications to Sorting

### 5.1 Sorting Using P Systems with Communication

We deal in this section with the problem of sorting a set of positive integers using P systems. It is not the purpose of this paper to treat *in extenso* the topic of sorting; it is presented here, mainly in order to be able to give in subsections 5.4 and 5.5 an illustration of how the concepts of dynamic P system and dynamic computation sequence can be used to solve "dynamic problems", problems for which the input data are not all available from the beginning, but are fed in a step-wise manner.

Still, a few words on the motivation for studying this kind of topic, and on the constraints our model uses, are necessary.

First, we want to sort by using *comparisons*, and not, for instance, counting. Second, we want to sort using P systems with communication *only*, that is, we will have no rewriting rules, and no methods based on rewritings.

If our sorting methods are based on comparisons, then we will be able to compare them with other methods, also based on comparisons. We mention here two of them: the classical (sequential) sorting algorithms, and the sorting networks, a model of computation able to perform (only) sorting, and which incorporates features of parallelism.

In general, comparing the new approach to computation proposed by P systems, with older, or classical models of computation, is an area where there is still much to be done. The problem of sorting seems appropriate for making a direct comparison possible. Also, there might be useful applications in the area concerned with constructing simulators for P systems: since the simulators use programming languages, which can be used directly to implement a sorting algorithm, then, the problem of finding one or several measures to account for the *cost* of a particular simulator can be addressed, and also different approaches to simulation could be compared.

### 5.2 Conventions

We will want to sort $n$ natural integers, say $X = \{x_1, \cdots, x_n\}$. We will represent an integer $x$ as a string $a^x$, i.e., the integer $x$ is represented in a membrane as $x$ occurrences of the same symbol, $a \in V$.

From this representation it immediately follows that every membrane can "hold" at most one integer, the multiplicity of $a$ inside it. It also follows that, if we want to sort $n$ items, we will need (at least) $n$ membranes, so the number of membranes depends on the dimension of the input for the sorting problem.

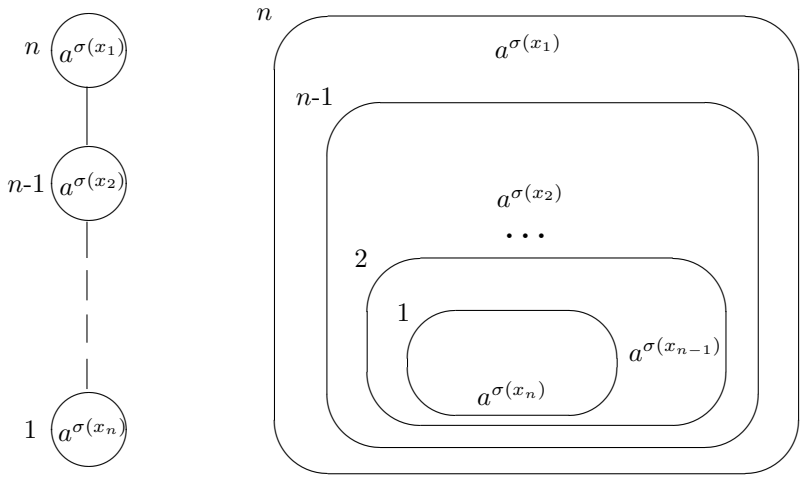



**Fig. 10**

We will use a structure of $n$ nested membranes. This nested structure allows for the codification of the result of sorting: if, for a given input sequence, $X = \{x_1, x_2, \cdots, x_n\}$, the solution to the sorting problem is the permutation of $n$ elements, $\sigma$, i.e., we have $\sigma(x_1) \leq \sigma(x_2) \leq \cdots \leq \sigma(x_n)$, then the nested membrane structure should, at the end of the sorting process, hold the integers in ascending order, from the outermost to the innermost membrane.

We say that a P system *has solved the sorting problem* for the input data $X = \{x_1, x_2, \cdots, x_n\}$, if it reaches such a configuration. (The configuration can be halting, or only stable over $\{a\}$.) Such a P system, and its associated tree structure are depicted in Fig. 10.

For the time being we do not deal with the problem of extracting the result of sorting.

### 5.3 The Comparator

Consider the following P system with symport/antiport:

$$C = (V, [_2\ [_1\ ]_1\ ]_2, \lambda, \lambda, E, R_1, \emptyset, i_0),$$

with

$$R_1 = \{(a, in; a, out) > (a, in)\}.$$

Its string description is the following expression in $ME(V)$:

$$C = [_2\ [_1\ a; a|\lambda; a\ ]_1\ ]_2.$$

Now suppose that membrane 1 contains $a^{x_1}$, and membrane 2 contains $a^{x_2}$, with $x_1, x_2 \in \mathbf{N}$, that is, consider the P system

$$C(x_1, x_2) = (V, [_2\ [_1\ ]_1\ ]_2, a^{x_1}, a^{x_2}, E, R_1, \emptyset, i_0),$$

whose string description in $ME(V)$ is:

$$C(x_1, x_2) = [_2|a^{x_2}|\ [_1\ a; a|a^{x_1}|\lambda; a\ ]_1|\ ]_2.$$

Let the rules act in their characteristic maximal parallel manner (feeding on input strings, if they find it): the antiport rule will make copies of $a$ to travel between regions 1 and 2, but always in pairs (if one $a$ gets out of region 1 into region 2, another $a$ will get from 2 into 1); the symport rule $(a, in)$, working in paralel with the antiport rule $(a, in; a, out)$, will have the following effect: if $x_2 > x_1$ then $x_2 - x_1$ copies of $a$ will travel from region 2 to region 1, so after one transition (computation step) we will have $a^{x_2}$ in membrane 1 and $a^{x_1}$ in membrane 2; if $x_2 < x_1$ the symport rule will have nothing to feed on, so we will have $a^{x_1}$ in membrane 1 and $a^{x_2}$ in membrane 2. In other words, we have achieved ordering $x_1$ and $x_2$ by placing the smallest of them into membrane 2 and the biggest into membrane 1.

We can formulate this alternatively in the following manner. Because of the *maximallity* of the parallelism of the rules, precisely $min(x_1, x_2)$ copies of $a$ will be acted upon by the antiport rule $(a, in; a, out)$, and recall that this is the highest priority rule. The second rule, $(a, in)$, can act on precisely $x_2 - min(x_1, x_2)$ occurrences of $a$ in membrane 2, and only if they are there. If $x_2 - min(x_1, x_2) > 0$ then $x_2 - min(x_1, x_2)$ occurrences of $a$ travel from membrane 2 inside membrane 1.

The achieved configuration is stable, because the only active rule remains the antiport rule $(a, in; a, out)$, which does not change the content of the membranes.

This means that, starting from the following initial configuration of the P system $C$, with objects inside the membranes

$$C(x_1, x_2) = [_1\ |a^{x_1}\ |\ [_2\ a; a|\ a^{x_2}\ |\ \lambda; a\ ]_2|\ ]_1,$$

after one internal computation step (maximal parallel use of rules) we obtain the configuration

$$CC(x_1, x_2) = [_1\ |a^{min(x_1,x_2)}\ |\ [_2\ a; a|\ a^{max(x_1,x_2)}\ |\ \lambda; a\ ]_2|\ ]_1,$$

which is *stable*, i.e., no further application of rules can change it, and which has solved the sorting problem for $\{x_1, x_2\}$.

Note that if we endow membrane 1 with the symport rule $(a, out)$ instead of $(a, in)$, we obtain the reverse order, the bigger number in the outer membrane.

In Figs. 11 and 12 we illustrate how our "comparator" works on input data $\{5, 3\}$.

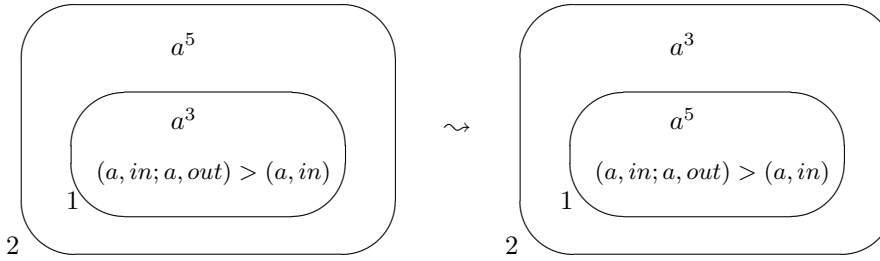



**Fig. 11.**

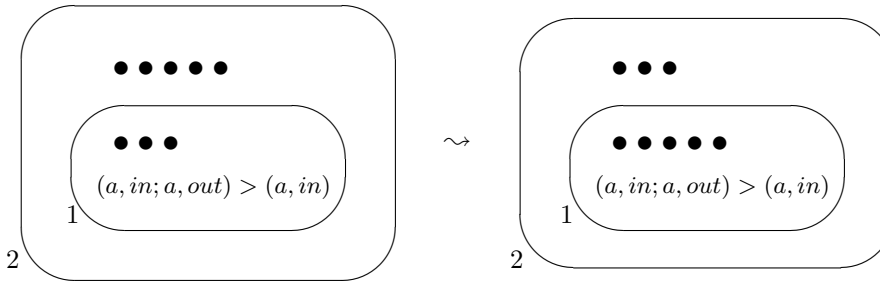



**Fig. 12.**

### 5.4 Dynamic Sorting with P Systems with Communication

Let $X$ be the set of positive integers which we intend to sort, i.e.,, the set of "input data" for the dynamic sorting mechanism which we will construct.

When writing $X = \{x_1, \cdots, x_n, \cdots\}$ we suppose that the integers of $X$ are "fed" into our dynamic sorting mechanism in the order given by the indices.

The dynamic P system which we will construct in the sequel, to solve (dynamically) the sorting problem for $X$, will be based on the comparator presented in subsection 5.3, only this time the rules will be active only in the presence of a promoter, $p \in V$. In other words, the comparator used in this section will be based on the following set of symport/antiport rules, with promoter $p$, and priority relation

$$C_p = \{(a, in; a, out)|_p > (a, in)|_p\}.$$

We will keep the same string notation for this pair of rules, as for the pair without promoter. Since the promoter itself will have to travel, the complete set of rules we will use is

$$R = \{(p, in) > (a, in; a, out)|_p > (a, in)|_p\}.$$

(The set $R$ is totally ordered only for convenience of the string notations which will follow. Except for the priority relation between the two rules which compose the comparator, and which is essential, the rule $(p, in)$ can be placed either as having the lowest priority, or outside the order.) Recall that promoters are allowed to leave a membrane only if the system has reached a stable configuration w.r.t. the rules the promoter has activated.

We construct now the contextual grammar which will generate the dynamic P system which solves the sorting problem for $X$. For the time being, we will consider $X$ finite.

Let $G_{sort} = (V \cup S, A, C, \phi)$ be the following *enriched bracketed* 5-*contextual grammar*:

– $V$ is the alphabet of objects, with $a, p \in V$;

– $S = \{[_i, ]_i \mid 0 \leq i \leq n-1\} \cup \{|\ ,\ ;\ \}$ is the set of separators;

– the set of axioms is a singleton, $A = \{\Pi_1\}$, with

$$\Pi_1 = [_1 | a^{x_1} | [_0 p | \lambda ]_0 | ]_1,$$

where $x_1 \in X$ (we will also use the notation $\Pi_1 = \Pi_1(x_1)$);

– the set of 5-contexts $C \subseteq ((V \cup S)^*)^5$ is

$$C = \{c(x) \mid x \in X \setminus \{x_1\}\},$$

where, for $i = 2, \cdots, n$, we have

$$c(x_i) = (\ [_i | a^{x_i} |\ ,\ p; a; a\ ,\ p|\ ,\ \lambda; a; \lambda\ ,\ |]_i).$$

– the 5-selection function $\phi : ((V \cup S)^*)^6 \to 2^C$ is defined by:

$$\phi(\lambda, [_{i-1}, |z|, [\beta]|, ]_{i-1}, \lambda) = \{c(x_i)\}$$

for $z \in a^*$, $\beta \in (V \cup S)^*$, $i = 2, \cdots, n$, and is $\emptyset$ on the rest of its domain.

Note that we have a strong relationship between the set of input data, $X$, and our grammar $G_{sort}$. One element of $X$, $x_1$, is used to build the axiom. The rest of them, $X \setminus \{x_1\}$, are used to construct the set of contexts $C$. A derivation in the grammar $G_{sort}$, which uses the context $c(x)$, accomplishes, among other things, the task of *feeding one more input data* – namely $x \in X$ – into our dynamic mechanism. In order not to use the same data again, we will have to further constrain the way in which derivations in $G_{sort}$ are applied.

Denote by $\Longrightarrow_x$ a derivation in $G_{sort}$ which uses context $c(x)$. If the first derivation, which is applied to the axiom, uses context $c(x)$, with $x \in X \setminus \{x_1\}$, then we denote this $x$ by $x_2$. In general, for every $i \geq 2$, if we have applied the succesive derivations $\Longrightarrow_{x_2}, \Longrightarrow_{x_3}, \cdots, \Longrightarrow_{x_i}$, then a next possible derivation is $\Longrightarrow_x$, with $x \in X \setminus \{x_1, \cdots, x_i\}$, and we label this next $x$ as $x_{i+1}$.

Note that this mechanism can be described in terms of contexts: each context $c(x) \in C$ can be used only once, and after being used we can imagine that it is erased from $C$; the selection function will pick up a new candidate for the next derivation from $C \setminus \{c(x)\}$.

If we work with this kind of constrained derivation, we can let $X$, and thus $C$, be infinite. In this case, $G_{sort}$ is a 5-contextual grammar with an infinite set of contexts. Moreover, we have:

**Lemma 6.** *The grammar $G_{sort}$ is deterministic.*

The notation we will use in the sequel for the strings which describe P systems obtained by derivations in this grammar, will keep track of the succesive derivations which have been applied, which is the same thing as keeping track of the input data from $X$ which have been used so far.

Let us see how the first derivation in $G_{sort}$ works, and also what we accomplish with it in terms of P systems.

Recall that our unique axiom is

$$\Pi_1 = \Pi_1(x_1) = [_1|a^{x_1}|[_0 p|\lambda]_0|]_1.$$

Only membrane 0 has rules, namely the rule $(p, in)$, but it cannot act since there is no $p$ in membrane 1. Therefore, there are no internal computations possible, so $\Pi_1 = C\Pi_1$. Applying the first derivation in $G_{sort}$, $\Pi_1(x_1) \Longrightarrow_{x_2} \Pi_2(x_1, x_2)$, we obtain

$$\Pi_2(x_1, x_2) = [_2|a^{x_2}|[_1 p; a; a|a^{x_1}|p|[_0 p|\lambda]_0|\lambda; a; \lambda]_1|]_2.$$

In terms of P systems, the derivation has accomplished the following:

– a new external membrane, with label 2, was generated, containing $a^{x_2}$;

– membrane 1 has been enriched with the set of rules $R$, and we have also fed a promoter $p$ into it.

The presence of the promoter makes the comparator in membrane 1 active, and internal computations are possible in $\Pi_2$. A first internal computation step (a transition) orders the integers $x_1, x_2$ by placing them in the apppropriate membrane. The configuration is now stable w.r.t. $\{a\}$, and also stable w.r.t. the set of rules $R$ in membrane 1, so the promoter $p$ can now travel, during the same transition, from membrane 1 into membrane 0, attracted by the rule $(p, in)$ of this membrane, and the system reaches a halting configuration, described by

$$C\Pi_2(x_1, x_2) = [_2\ |a^{min(x_1,x_2)}\ |\ [_1\ p; a; a|\ a^{max(x_1,x_2)}\ |[_0 p|p|\lambda]_0\ |\ \lambda; a; \lambda\ ]_1|\ ]_2.$$

Consider now the dynamic P system with halting computations associated to $G_{sort}$,

$$\Pi(G_{sort}) = \{\Pi_n(x_1, \cdots, x_n) \mid n \geq 1\},$$

whose terms are given by the dynamic computation sequence

$$\begin{array}{lll} \Pi_1(x_1) \Longrightarrow_{x_2} \Pi_2(x_1, x_2) & \rightsquigarrow C\Pi_2(x_1, x_2) \\ \quad \Longrightarrow_{x_3} \Pi_3(x_1, x_2, x_3) & \rightsquigarrow C\Pi_3(x_1, x_2, x_3) \\ \quad \cdots \\ \quad \Longrightarrow_{x_n} \Pi_n(x_1, \cdots, x_n) & \rightsquigarrow C\Pi_n(x_1, \cdots, x_n)\ \cdots \end{array}$$

where the internal computation step $\Pi_i \rightsquigarrow C\Pi_i$ consists of all possible internal computations (all transitions) till the configuration $C\Pi_i$ is a halting one.

We have the following result:

**Theorem 1.** *The dynamic P system with halting computations $\Pi(G_{sort})$, generated by the enriched bracketed 5-contextual grammar $G_{sort}$, solves the sorting problem for the dynamic input sequence $X = \{x_1, x_2, \cdots, x_n, \cdots\}$.*

*More precisely, for every $i \geq 1$, the system $C\Pi_i(x_1, \cdots, x_i)$ is a halting configuration of $\Pi_i(x_1, \cdots, x_i)$, and has solved the sorting problem for $\{x_1, x_2, \cdots, x_i\}$ (in the sense that the integers are placed in the membranes in increasing order, from the external membrane $i$ to membrane 1).*

*The average time is $\mathcal{O}(n^2)$ for input dimension $|X| = n$.*

*Proof.* By induction on $n$. For $n = 1$, the axiom $\Pi_1(x_1) = C\Pi_1(x_1)$ is a halting configuration, and it solves the sorting problem for the singleton $\{x_1\}$.

We have seen how the system $\Pi_2(x_1, x_2)$ solves the sorting problem for $\{x_1, x_2\}$, and reaches the halting configuration $C\Pi_2(x_1, x_2)$, in which no rules are applicable because the promoter has entered membrane 0.

Let the induction hypothesis be that $C\Pi_i(x_1, \cdots, x_i)$ has solved the sorting problem for $\{x_1, x_2, \cdots, x_i\}$, and is a halting configuration of $\Pi_i(x_1, \cdots, x_i)$. This means that the $i$ nested membranes of the P system contain the naturals in increasing order, from the skin membrane towards membrane 1, and all the promoters used so far have travelled into membrane 0. We can suppose that $x_1 \leq \cdots \leq x_i$, and thus the string description of $C\Pi_i$ is:

$$C\Pi_i = [_i|a^{x_1}|[_{i-1}p; a; a| \cdots [_0p|p^{i-1}|\lambda]_0|\lambda; a; \lambda]_{i-1}|]_i.$$

We apply now the derivation $\Longrightarrow_{x_{i+1}}$ in $G_{sort}$, and obtain the P system

$$\Pi_{i+1} = [_{i+1}|a^{x_{i+1}}|[_ip; a; a|a^{x_1}|p|[_{i-1}p; a; a| \cdots [_0p|p^{i-1}|\lambda]_0|\lambda; a; \lambda]_{i-1}|\lambda; a; \lambda]_i|]_{i+1}.$$

The effect of the derivation is, in terms of P systems:

– a new external membrane was generated, labelled $i + 1$, and containing $a^{x_{i+1}}$;

– the set of rules $R$ was added to membrane $i$, together with a promoter $p$, which activates them.

After activating the comparator rules $C_p$ which do the sorting between membranes $i$ and $i + 1$, the promoter travels to the next inner membrane, $i - 1$, activating its comparator $C_p$, and so on. At some point, the newly introduced integer will find its appropriate place in one of the membranes. When this happens, the obtained system has attained a configuration which is stable w.r.t. $\{a\}$, and has solved the sorting problem. Further on, $p$ will continue traveling, without altering the stability w.r.t. $\{a\}$ of the succesive configurations, until it drops to membrane 0, and the configuration reached is now halting.

The mechanism is very similar to the classical insertion sort, and the estimate of its complexity comes from this similarity, where we have counted 1 for each transition during the internal computations, and again as 1 the cost of each derivation in $G_{sort}$. □

### 5.5 A Second Dynamic P System for Sorting

We present here another dynamic P system with halting computations which solves the sorting problem. This alternative model is based on an enriched bracketed insertion grammar, working also on $ME(V)/\sim$, so we will have a concrete example of such a type of grammar.

The sorting mechanism is similar to the one used by $G_{sort}$, and it is based on the same comparator $C_p$; the complete set of rules used by an active membrane will be

$$Q = \{(a, in; a, out)|_p > (a, in)|_p > (p, out)\}.$$

Let $G_{sort2} = (V \cup S, A, P)$ be the following *enriched bracketed insertion grammar*:

– $V$ is the alphabet of objects, with $a, p \in V$;

– $S = \{[_i, ]_i \mid 0 \leq i \leq n-1\} \cup \{|\ ,;\ \}$ is the set of separators;

– The set of axioms is a singleton, $A = \{\Pi_1\}$, with

$$\Pi_1 = \Pi_1(x_1) = [_1 \lambda | a^{x_1} | p]_1,$$

where $x_1 \in X$. Its unique membrane, 1, contains $a^{x_1}$, and the unique rule $(p, out)$. The configuration is a halting one, and $\Pi_1(x_1) = C\Pi_1(x_1)$.

– The set of insertion rules $P = \{p_i \mid 1 \leq i \leq n\}$ is in bijective correspondence with the subset of input data $X \backslash \{x_1\}$. More precisely, for each $i \geq 1$, the insertion rule $p_i$ is a triple

$$p_i = (\ [_i w|\beta|\ ,\ [_{i+1} a; a; \lambda | a^{x_{i+1}} | p | p; \lambda; a]_{i+1}\ ,\ z]_i\ )$$

with $\beta \in a^*$, and $w, z \in (V \cup S)^*$ such that $(w, z) = (a; a; \lambda, p; \lambda; a)$ or $(w, z) = (\lambda, p)$.

Let us see how the first derivation in $G_{sort2}$ works, $\Pi_1(x_1) \Longrightarrow_{p_1} \Pi_2(x_1, x_2)$. It can use only the production

$$p_1 = ([_1 \lambda | a^{x_1}|, [_2 a; a; \lambda | a^{x_2} | p | p; \lambda; a]_2, p]_1),$$

and by applying it we obtain

$$\Pi_2(x_1, x_2) = [_1 \lambda | a^{x_1} | [_2 a; a; \lambda | a^{x_2} | p | p; \lambda; a]_2 p]_1.$$

In terms of P systems, the derivation has accomplished the following:

– it has generated an elementary membrane, labelled 2, inside membrane 1;

– the newly generated membrane contains $a^{x_2}$, rules $Q$, and the promoter $p$.

In $\Pi_2(x_1, x_2)$ internal computations are possible: the presence of the promoter $p$ in membrane 2 makes the comparator rules active, and they will order $x_1$ and $x_2$ in their respective membranes. After one transition, the system is stable w.r.t. $\{a\}$, and also w.r.t. the rules $C_p$, so the promoter can travel. It will be sent to membrane 1 by the rule $(p, out)$ of membrane 2, and during the next transition, it will be sent into the environment by the rule $(p, out)$ of membrane 1. The halting configuration $C\Pi_2(x_1, x_2)$ is reached, where

$$C\Pi_2(x_1, x_2) = [_1 \lambda | a^{min(x_1, x_2)} | [_2 a; a; \lambda | a^{max(x_1, x_2)} | p; \lambda; a]_2 | p]_1.$$

Note that the *only* insertion rule applicable to $C\Pi_2(x_1, x_2)$, and thus the one used by the next derivation, is

$$p_2 = ([_2 a; a; \lambda | a^{max(x_1,x_2)}| \ , \ [_3 a; a; \lambda | a^{x3} | p | p; \lambda; a]_3 \ , \ p; \lambda; a]_2),$$

and the next derivation produces the P system

$$\Pi_3(x_1, x_2, x_3) =$$
$$= [_1 \lambda | a^{min(x_1,x_2)} | [_2 a; a; \lambda | a^{max(x_1,x_2)} | [_3 a; a; \lambda | a^{x3} | p | p; \lambda; a]_3 p; \lambda; a]_2 | p]_1.$$

We have the same type of correspondence between the set of input data, $X$, and the set of insertion rules of $G_{sort2}$, as we had for $G_{sort}$.

In particular, if $X$ is infinite, grammar $G_{sort2}$ is an insertion grammar with an infinite set of insertion strings. We also have:

**Lemma 7.** *The grammar $G_{sort2}$ is deterministic.*

Consider now the dynamic P system with halting computations associated to $G_{sort2}$,

$$\Pi(G_{sort2}) = \{\Pi_n(x_1, \cdots, x_n) \mid n \geq 1\},$$

whose terms are given by the dynamic computation sequence

$$\begin{array}{lll} \Pi_1(x_1) \Longrightarrow_{p_1} & \Pi_2(x_1, x_2) & \rightsquigarrow C\Pi_2(x_1, x_2) \\ \Longrightarrow_{p_2} & \Pi_3(x_1, x_2, x_3) & \rightsquigarrow C\Pi_3(x_1, x_2, x_3) \\ \cdots & & \\ \Longrightarrow_{p_{n-1}} & \Pi_n(x_1, \cdots, x_n) & \rightsquigarrow C\Pi_n(x_1, \cdots, x_n), \cdots \end{array}$$

where the internal computation step $\Pi_i \rightsquigarrow C\Pi_i$ is of the halting type.

We have the following result:

**Theorem 2.** *The dynamic P system with halting computations $\Pi(G_{sort2})$, generated by the enriched bracketed insertion grammar $G_{sort2}$, solves the sorting problem for the dynamic input sequence $X = \{x_1, x_2, \cdots, x_n, \cdots\}$.*

*More precisely, for every $i \geq 1$, the system $C\Pi_i(x_1, \cdots, x_i)$ is a halting configuration of $\Pi_i(x_1, \cdots, x_i)$, and has solved the sorting problem for $\{x_1, x_2, \cdots, x_i\}$ (in the sense that the integers are placed in the membranes in increasing order, from membrane 1 to membrane $i$).*

*The average performance is $\mathcal{O}(n^2)$ for $|X| = n$.*

The proof is again based on an induction argument, and is similar to that of Theorem 1.

Note that the mechanisms used by the two grammars are symmetrical. While $G_{sort}$ adds the new data in new external membranes, and makes the promoter travel "from outside", activating one-by-one the comparators of each membrane, and is finally collected into membrane 0, grammar $G_{sort2}$ adds the new data in new internal membranes, and makes the promoter travel "from inside", activating the rules, and is finally ejected into the environment.

### 5.6 Performance

Note that the sorting mechanisms described above are both similar to that of insertion sort, and thus the low performance.

We can make better use of the parallelism of the internal computations of a P systems. Recall that we have used only one letter $a$ of our alphabet of objects $V$, to sort only one sequence. With a dynamic P system similar to the ones described, we could sort simultaneously $card(V) - 1$ sequences, of course, by using $2 \times (card(V) - 1)$ rules.

We know of only one approach to the sorting problem by means of P systems, namely, the Bead-Sort algorithm presented in [1]. It also uses only symport/antiport rules, and the mechanism of "beads sliding to their appropriate places" (on rods) is similar to our travel of occurrences of $a$ through membranes.

In order to achieve a certain configuration (in which the actual sorting process is finished, and which is similar to our configuration of nested membranes), the Bead-Sort algorithm [1], implemented with P systems, uses $n \times m$ membranes, with communication rules, where $n$ is the dimension of the input data set, and $m = |max\{x_1, \cdots, x_n\}|$ (the number of "rods"), is the length of the maximal input data. So, it is constrained both by the fixed size, $n$, of the cardinality of the set of input data, and by the length $m$ of its maximal element.

We use only $n$ membranes, to achieve an easily "readable" configuration, and the dynamic nature of the P system allows for growing the cardinality of the set of input data.

More precisely, in our model, compared with the model in [1]:

– we have eliminated the constraint related to the maximal size of input data (the $m$), by using the comparator $C_p$; the "price" we have paid comes in the form of concepts (features) which have not been used so far, such as using priority relations on communication rules, and notions of stability for P systems;

– we have added more flexibility to the constraint based on the dimension $n$ of the input data set, by using *dynamic* P systems.

In both approaches, the input has to be afterwards extracted, and we do not take into account here the cost of the extraction process, since we have not implemented (yet) this feature into our model. But we feel confident that this can be readily done, and at a lower cost.

## 6 Concluding Remarks and Open Problems

In the first part of the paper we have proposed a generative approach to P systems. We have introduced a new class of contextual grammars, the enriched bracketed contextual grammars, which are generative devices for families of P systems with symport/antiport rules. In general, the enriched bracketed contextual grammars give rise to new hierarchies of P systems with communication, whose computational powers need to be investigated, and also compared. The definitions in subsections 3.3, 3.4, and 3.5 are only the beginning of what should be a systematic enquiry about the families of P systems one can obtain, and whose computational powers need to be investigated.

Starting, for instance, from the discussions in subsections 3.3 and 3.4, one can easily construct a set of expressions, and a particular type of enriched bracketed contextual grammars acting on them, to obtain a generative device for P systems *with at most one symport/antiport rule per membrane*. What can be computed by means of such systems? (Or, in other words, what can be computed by these particular enriched bracketed contextual grammars?) Since for rewriting P systems (hence with string-objects) it is known (see [12]) that universality can be achieved by systems with three rules in each region, it is meaningful to ask if a similar normal form result is valid for systems with symport/antiport. And also, can the number of rules of each region be decreased to one? This last question is equivalent to: what can be computed by the above-mentioned particular enriched bracketed contextual grammars?

As pointed out in subsection 3.6, and also in Section 4, the generative approach we have proposed, based on (possibly new types of) contextual garmmars, is very flexible, in the sense that it can be easily used as a model to construct grammars generating other kinds of P systems. In [2] for instance, it has been used to generate P systems with one rewriting rule per membrane. All the open problems, and further research topics, we have proposed in subsection 3.6 for P systems with communication, can now be formulated for other kinds of P systems as well.

We hope that our generative approach has also proved the power of the contextual generative mechanism, and the fact that the area of contextual grammars is a rather fruitful and active field of investigation.

We have also introduced the concept of dynamic P system and of dynamic computation sequence, in a very general form.

In section 4 we have formulated some further research topics related to the investigation of other types of dynamic computation sequences, based on the similarity with CD grammar systems. Investigating more complicated "communication protocols" between the grammar derivations, and the internal computation steps, is also an area for further research.

We also want to point out that, what we have proposed as a "generative" device, can be also used as a "descriptive" tool. Take, for instance, the P systems with enhanced membrane handling introduced in [6]: they are enriched with rules describing the behaviour of the membranes (appart from computations done by rewriting and by communication), rules acting as part of the computation process. If the behaviour of the membranes can be modelled with a contextual grammar mechanisms, as in the present approach, then the whole "computational" behavior of such a system can be described by a dynamic computation sequence, maybe of a more general type than that introduced in this paper.

We have also introduced a model of sorting with symport/antiport rules. We have presented here only two *dynamic* versions of sorting, for exemplifying purposes. But the whole topic of sorting, especially the *static* versions which can be constructed based on the same comparator, is worth of further investigation (and is presently under such investigation).

Moreover, we have introduced new concepts for the behaviour of P systems, like stable configuration, and the weaker notions of stability w.r.t. a subalphabet, or w.r.t. a subset of rules. Also, we have considered a priority relation for sets of communication rules. We believe that these notions can prove their usefulness in other settings as well.

**Acknowledgements**

The authors acknowledge fruitful discussions with Rudolf Freund and Pierluigi Frisco on the topic of sorting.

## References

1. J.J. Arulanandham: Implementing Bead-Sort with P Systems, in *Unconventional Models of Computation 2002* (C.S. Calude, M.J. Dineen, F. Peper, eds.), *Lecture Notes in Computer Science*, 2509, Springer-Verlag, Heidelberg, 2002, 115–125
2. G. Bel-Enguix, M. Cavaliere, R. Ceterchi, R. Gramatovici, C. Martín-Vide: An Application of Dynamic P Systems: Generating Context-Free Languages, this volume
3. R. Ceterchi, C. Martín-Vide: Generating P Systems with Contextual Grammars, *Pre-proceedings of the Workshop on Membrane Computing*, Curtea de Argeş, 2002, 119–144
4. E. Csuhaj-Varjú, J. Dassow, J. Kelemen, G. Păun: *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach Science Publ., London, 1994
5. R. Freund, M. Oswald: P systems with activated/prohibited membrane channels, paper presented at WMC 2002, Curtea de Argeş, Romania
6. M. Margenstern, C. Martín-Vide, G. Păun: Computing with Membranes: Variants with Enhanced Membrane Handling, *Proceedings of 7th Intern. Meeting on DNA Based Computers* (N. Jonoska, N.C. Seeman, eds.), Tampa, Florida, USA, 2001, 53–62
7. C. Martín-Vide, G. Păun: Structured Contextual Grammars, *Grammars,* 1, 1 (1998), 33–55.
8. A. Păun, G. Păun: The power of Communication: P Systems with Symport/Antiport, *New Generation Computers*, 20, 3(2002), 295–306
9. G. Păun: *Marcus Contextual Grammars,* Kluwer, Dordrecht, 1997
10. G. Păun: Computing with Membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for CS-TUCS Report* No. 208, 1998
11. G. Păun: *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, Heidelberg, 2002
12. C. Zandron, C. Ferretti, G. Mauri: Two Normal Forms for Rewriting P Systems, *Proc. Third. Intern. Conf. on Universal Machines and Computations*, Chişinău, Moldova, 2001 (M. Margenstern, Y. Rogozhin, eds.), *Lecture Notes in Computer Science*, 2055, Springer, 2001, 153–164
13. *http://psystems.disco.unimib.it/*

# Membrane Systems and Distributed Computing

Gabriel Ciobanu*, Rahul Desai, and Akash Kumar

National University of Singapore, School of Computing
Department of Computer Science
gabriel@{comp.nus.edu.sg,info.uaic.ro}

**Abstract.** This paper presents membrane systems as an appropriate model for distributed computing, an efficient and natural environment to present the fundamental distributed algorithms. We support the idea that P systems can become a primary model for distributed computing, particularly for message-passing algorithms. We present the core theory, the fundamental algorithms and problems in distributed computing. We focus on an example describing an immune response system against virus attacks. The example is implemented using a P system library created by the authors to simulate the main functions of a P system, and an MPI program that takes advantage of the highly parallel features provided by the model. The program uses distributed leader election and synchronization algorithms.

## 1 Membrane and Molecular Computing

Formal language theory has been used as a basis for developing theoretical computational models related to DNA sequences and molecular processes. These developments reveal some theoretical facets of *molecular computing* related mainly to the computational power of the new systems, generative capability, complexity, and universality.

Membrane computing is based on *membrane systems* or P systems, a new class of distributed and parallel computing devices introduced in [7]. The approach is based on hierarchical systems: finite cell-structures consisting of cell-membranes embedded in a main membrane called the *skin*. The membranes determine *regions* where *objects*, elements of a finite set, and evolution rules can be placed. The objects evolve according to given *rules* associated with a region. Objects may also move between regions. A *computation* starts from an initial configuration of the system, and terminates when no further rule can be applied. A software simulator of membrane systems is presented in [3].

A membrane structure is usually represented by a Venn diagram, and it can be mathematically represented by a tree or by a string of matching parentheses. Hierarchical systems are well-known structures in computer science, and the notion of computation based on evolution rules is common. The interpretation of the computation is rather new: the result of a computation is a multiset of objects collected in the output cell or sent out of the system. The behaviour of the

* Corresponding author

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 187–202, 2003.

whole system is obtained by combining the resulting multisets, or considering the multiplicity of objects present in a specific output membrane of a final configuration. Păun has introduced initially three alternatives to look at membrane systems. Starting from these approaches, several variants were considered. Each of these variants has been shown to generate recursively enumerable sets. All these results emphasize a new computing paradigm inspired by a basic function of biomembranes.

## 2 The Distributed Nature of the Membrane Systems

An interesting aspect of the membrane systems is that they are distributed and parallel computing devices. To complement the overwhelming majority of researches in this area dealing with computability results, this paper aims to show that the membrane systems represent an appropriate model for distributed computing. We emphasize on the algorithmic aspects related to the distributed systems computational power provided by membrane systems. The algorithms are mainly presented in [2]. In this paper we focus on an example describing an immune response system against virus attacks. The example uses various distributed algorithms, and it is implemented using a P system library and an MPI program emphasizing the highly parallel features provided by the model.

Another approach presenting the membrane systems as distributed and parallel computing devices is described in [4], where a new version of P systems called Client–Server P systems is introduced. The new version is devoted to the interaction between components and is similar to the network client-server model. The Client–Server P systems are based mainly on the power of communication between membranes, and they have the same expressive power as Turing machines.

The link between membrane systems and distributed and parallel computing is not difficult to comprehend. If we associate each membrane with a host on a network, then the membrane containing a few such membranes is congruent to a subnet, and subsequently their parent membranes represent larger networks. Finally, the skin membrane could represent the World Wide Web. You can imagine that routing in Internet is similarly to message passing within a P system. Specifically, membranes within the same membrane can directly communicate with each other (by sending objects within the membrane), while if two membranes in different parent membranes need to pass objects to each other, then this scenario is congruent to sending the packet to a router which connects the two networks. Having established the basic premise of the paper, we will conclusively show how P systems provide a natural and efficient representation of distributed systems.

In the context of parallel and distributed systems, the algorithmic issues studied in the sequential model require a fundamental rethinking. In parallel systems, a problem is solved by a tightly-coupled set of processors, while in distributed systems it is solved by a set of communicating (asynchronous) processes. From the viewpoint of the theory of distributed computing, which is restricted to the

Message Passing and Shared Memory model, the P systems model provides a different perspective, which is natural and easy to relate to. In P systems the process of passing objects is similar to message passing; moreover, membranes in the same biological system could have access to the same DNA, or could have access to the same blood stream, making it possible to relate the model to a Shared Memory system as well. Both these extensions are simplistic and natural, and hence it is not difficult to adapt the already existing theory of distributed computing to the P systems model, something that this paper aims to achieve.

We emphasize on the algorithmic aspects related to the distributed systems computational power provided by membrane systems. In membrane systems the process of passing objects through membranes in both directions is similar to message passing. We consider a system of communicating membranes with antiport carriers, and the main meaning regarding this choice is that the membranes *send* and *receive* information. We present some basic algorithms of distributed computing, starting with algorithms for broadcast, convergecast, flooding, leader election, mutual exclusion in distributed systems, the fault tolerant systems and the consensus problem.

We present the fundamentals of distributed computing, starting with algorithms for broadcast, convergecast, the leader election problem, the mutual exclusion problem in a distributed environment, and finally the consensus problem and fault tolerance. Some of the presented algorithms are used in an example describing an immune response system against virus attacks.

## 3 Basic Algorithms in P Systems

The field of distributed computing is notoriously difficult, mainly due to uncertainties introduced by limited local knowledge, asynchrony, and failures. The fundamental issues underlying the design of distributed systems are related to communication, coordination, synchronization and fault tolerance. Mastering fundamental algorithmic ideas and techniques, someone is able to design correct distributed systems and applications. We present here some basic algorithms over communicating membrane systems, algorithms representing the core theory of distributed computing. For more information and notation about fundamental distributed algorithms, see [1].

Collecting and dispersing information is central to any system. Even though P systems share the same DNA, local information often has to be passed around in the system. This is where the message passing model becomes relevant. Two basic algorithms in any message passing model are *broadcast* and *convergecast*. Another common algorithm discussed in a message passing model is *flooding*. This algorithm constructs a spanning tree when a graph is given. P systems themselves essentially have a spanning tree structure. Each membrane of the tree has exactly one parent, except the skin membrane which represents the root.

**Broadcast:** Consider a system in which a membrane $m_i$ has to send some object to all the membranes of the system. Clearly, we have two cases of broadcast here - one in which $m_i$ is the skin membrane or the root node, and secondly when $m_i$ is any node. It is not difficult to see that the second case is a generalization of the first one. We shall start with the algorithm for the simple case (when $m_i$ is the skin membrane), for easy understanding.

Very often in a distributed computing model, the root node has to broadcast a certain message, say $M$. Here is the prose description of the algorithm. The skin membrane, $m_s$ first sends the message to all its children. Upon receiving a message from its parent, the membrane sends the message to all its children, if any. Here is the formal pseudocode for this algorithm.

**Algorithm: Skin Membrane Broadcast**

Initially $M$ is in transit from $m_s$ to all its children.

Code for $m_s$:

1. Upon receiving no message:
2. terminate

Code for $m_i, 0 \leq i \leq n-1, i \neq s$:

3. Upon receiving $M$ from its parent:
4. send $M$ to all its children
5. terminate

A skin broadcast algorithm for P systems has a message complexity of $n-1$ and time complexity $l$, when $l$ levels of membranes are present.

*Message Complexity:* As evident from the above algorithm, the message is communicated from a parent membrane to a child membrane exactly once. The algorithm terminates after sending $M$ once to all its children. Thus, the total number of messages passed is equal to the number of edges in the spanning tree structure. Since, the number of edges in a spanning tree with $n$ nodes is $n-1$, we obtain the result that $n-1$ messages are passed in a P system with $n$ membranes. Therefore, the message complexity of this algorithm is $O(n)$.

*Time Complexity:* In every admissible execution of the skin broadcast algorithm, every membrane at level $l$, i.e. at a distance of $l$ edges from the root node in the spanning tree, receives the message $M$ in $l$ time. Thus a P system with $l$ levels of membranes will have a time complexity of $l$. This corresponds to a depth of $l$ in a spanning tree configuration. In the worst case, when the spanning tree is a chain, there can be at most $n-1$ levels for a system with $n$ membranes. This shows that the time complexity of any P system for skin broadcast is $O(n)$.

### 3.1 Generalised Broadcast

Having seen broadcast for the skin membrane, we shall now move on to a more general broadcast in which any membrane can broadcast a message. Each membrane $m_i$ which needs to broadcast sends the object $M$ to its parent and all

children (if any). A membrane $m_j$, upon receiving a message from its parent, sends it to its children. If it receives the message from its child, then it sends the message to all its other children, and parent. The algorithm is given as follows.

**Algorithm: Generalised Broadcast**

Say, membrane $m_a$, $0 \leq a \leq n-1$ needs to send the message to all the membranes in the system.

Code for $m_a$:

1. if $(a \neq s)$
2. send $M$ to its parent
3. send $M$ to all children

Code for $m_i$, $0 \leq i \leq n-1, i \neq a$:

4. Upon receiving $M$ from its parent:
5. send $M$ to all children
6. Upon receiving $M$ from its child:
7. if$(i \neq s)$
8. send $M$ to its parent
9. send $M$ to all children

A generalized broadcast algorithm for P systems has a message complexity of $n-1$ and a time complexity of $l+k$, when $l$ levels of membranes are present and the membrane at $k^{th}$ level broadcast. Message complexity is similar to the skin broadcast algorithm, and it is $O(n)$. A P system with $l$ levels of membranes will have a worst case time complexity of $2 \times l$. This means that the time complexity for generalized broadcast is $O(n)$. More details are presented in [2].

### 3.2 Convergecast

The broadcast problem mentioned above aims at dispersing information held by a membrane to other membranes of the system. Convergecast, on the contrary, aims at collecting information from all the membranes of the system to the skin membrane. Many variants of the problem are available, for example, forwarding the sum of all the values held by membranes, or forwarding the maximum value, etc. In a general convergecast algorithm, instead of the result of a particular operation, all the values are forwarded. In a generalized convergecast variant, the size of message can increase as the message progresses to the skin membrane. For simplicity we shall consider the algorithm of forwarding the sum of all the values held by membranes.

As it can be seen, unlike broadcast which is initiated by the membrane that wishes to disseminate information, convergecast is initiated by the *leaves*, i.e. membranes which contain no inner membranes. This algorithm is recursive, and requires each membrane $m_i$ to forward the sum of values held by its membrane. In other words, the sum of the subtree rooted at it. A membrane collects all

the values held by its inner membranes, and computes the sum including its own values. This sum $s_i$ is then forwarded to its parent membrane. Clearly, each membrane has to receive a sum from each of its children before it can forward the sum to its parent. The pseudocode for the algorithm is given below.

**Algorithm: Convergecast**

Code for leaf membranes:

1. Starts the algorithm by sending its value $x_i$ to its parent.

Code for non-leaf membranes $m_i$ with $k$ children:

2. Waits to receive messages containing sums $s_{i_1}, s_{i_2}, \ldots, s_{i_k}$ from its children $m_{i_1}, m_{i_2}, \ldots, m_{i_k}$.
3. Computes $s_i = x_i + s_{i_1} + \ldots + s_{i_k}$
4. if $(i \neq s)$
5. Sends $s_i$ to its parent.

The analysis of this algorithm is analogous to the skin broadcast algorithm, since the only difference in the two is the direction of message flow. As for the skin broadcast algorithm, the message complexity of the algorithm is $n-1$. The time complexity of the algorithm is $O(n)$, since at most $n-1$ levels may be present in a P system with $n$ membranes. Therefore there is a convergecast algorithm for P systems with message complexity $n-1$ and time complexity $l$, when $l$ levels of membranes are present.

## 4 Leader Election in P Systems

The existence of a leader in a P system can often simplify the task of coordination among the membranes. It might often be useful to have a leader (other than the skin-membrane as the default). It might also be the case that the criterion for leadership may not always be met by the skin-membrane. The leadership election problem, generally refers to the general class of symmetry breaking problems. The most general variant of it requires exactly one node from a system of many initially similar nodes to declare itself the *leader*, while the others recognize the leader and declare themselves *not-elected*.

**Theorem 1.** *It is impossible to solve the leadership election problem in a system where the membranes are anonymous.*

The idea behind this impossibility result is that the symmetry between the membranes can be maintained forever if the membranes are anonymous (they are very similar). Without some initial asymmetry provided by unique identifiers, symmetry cannot be broken and it is impossible to elect a single leader: if one membrane is elected, then so are all the membranes. Therefore, we assume that every membrane in the system has one unique identifier *id*. An algorithm is said to be *uniform*, if it does not depend on the number of membranes. And conversely, *non-uniform* algorithms rely on the knowledge of the number of membranes.

### 4.1 A Simple Leader Election Algorithm

The most straightforward way to solve the problem is that every membrane sends a message with the maximum *id* among all its children (and itself) to its parent, and waits for a response from its parent. The skin membrane in turn, would reply to all its children with a message containing the maximum *id* that it received. Ultimately, one membrane (which receives its own id back) will be elected leader.

**Algorithm: Leader Election**

Initially: *elected* = false, *children* = set of children membrane,
and *parent* = the parent membrane.
For every membrane $m_i$

1. If *children* = empty, send *id* to *parent*
2. Upon receiving $id_j$ from all children,
   $winner = max(id_1, id_2, \ldots, id_n, id)$
3. If *parent* $\neq$ *null*
   then send *winner* to *parent*
   else (it is the skin membrane)
   send *winner* as *leader* to all children
4. Upon receiving message *leader* from parent
   if *leader* = *id* then *elected* = true
5. If *children* = empty, terminate
   else send *leader* to all children and terminate.

The message complexity of the above algorithm is $O(n)$, since every link between the parent and the child is used to exchange 2 messages. In a system with $n$ membranes, there are $n-1$ such links. Thus, the message complexity is $O(n)$. The leadership algorithms in asynchronous rings have a lower bound of $O(n\,logn)$. Whereas, in the synchronous case, a message complexity of $O(n)$ can be achieved at the cost of the time-complexity [1]. Generally, the P systems are structured like a tree, already providing a sufficient edge to break-symmetry as compared to a ring, where every substructure of the ring is symmetrical.

## 5 Mutual Exclusion in Shared Memory

*Shared memory* is another major communication model and we shall see how P systems can be used effectively to model this as well. In a shared memory, processors communicate via a common memory area that contains a set of *shared variables*, which are also referred to as *registers*. In natural computing using P systems, as said above, membranes have access to the same blood stream and mutual exclusion can thus be easily modelled.

### 5.1 Formal Model of Shared Memory Systems

Before we proceed to understand mutual exclusion algorithms, we need to define the formal model for a shared memory system. We assume we have a system with $n$ membranes $m_0, m_1, \ldots, m_{n-1}$ and $m$ registers or shared variables $R_0, \ldots, R_{m-1}$. Each register (shared variable) has a *type* which can specify the values which a register can take, the operations that can be performed on it, the value returned by each operation, and the updated value of the register as a result of the operation. Beside this, an initial value for the register has to be specified. Another important distinction in shared memory systems comes when analyzing algorithms. Unlike in object passing models, object complexity is meaningless. On the other hand, *space complexity* becomes relevant in this model. Space complexity can be measured in two ways: number of registers used, and number of distinct values the register can take. Measuring time complexity of shared memory algorithms is still a current research area and we only focus on whether the number of steps in the worst case running of the algorithm is infinite, finite, or bounded.

### 5.2 The Mutual Exclusion Problem

The *mutual exclusion* problem is one where different membranes need access to a shared resource that cannot be used simultaneously. Some relevant terms in the section are given below:

- *Critical Section:* Code segment that has to be executed by at most one membrane at any time.
- *Deadlock:* A situation in which when one or more membranes are trying to gain access to a critical section, and none of them succeeds.
- *Lockout:* When lockout occurs, a membrane trying to enter its critical section never succeeds.

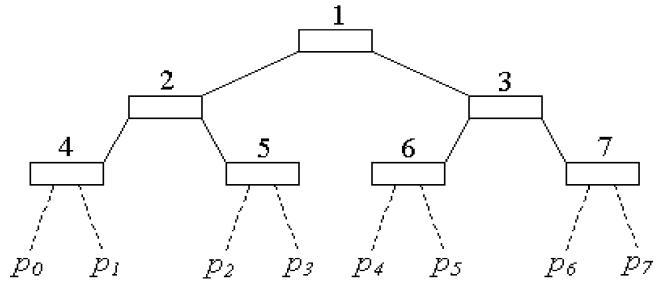
A membrane might need to execute some additional code segments before and after the critical section. This is to ensure mutual exclusion. The relevant sections of the code are:

- *Entry:* Code section where the membrane prepares to enter critical section.
- *Critical Section:* Code section which has to be executed exclusively.
- *Exit:* Code section executed when a membrane leaves the critical section.
- *Remainder:* Remainder of the code.

The desired properties are mutual exclusion, no deadlock and no lockout. *Mutual Exclusion* is achieved when in every configuration of every execution at most one membrane gets access to critical section. *No Deadlock* is achieved in every admissible execution, when membranes are in the *entry* section, at a later stage, a membrane is definitely in the critical section. *No Lockout* is achieved when in every admissible execution, a membrane trying to enter the critical section, eventually gets an entry.

### 5.3 Achieving Mutual Exclusion

The *test-and-set* and *read-modify-write* are powerful primitives used to achieve mutual exclusion. Another commonly known algorithm is the *Bakery algorithm* which uses *read/write* registers [1]. An appropriate algorithm for P systems would be the *tournament algorithm*. The conventional tournament algorithm can be modified to suit P systems. The tournament algorithm is a bounded mutual algorithm for $n$ processors. It is based on selecting one among two processors at every stage, and thus selecting one among $n$ processors in $\lceil log_2 n \rceil$ stages. The algorithm is recursive and every processor that succeeds at a stage climbs up the binary tree. The processor reaching the root gains entry to the critical section. An example with 8 processors is presented below:




Membranes within a parent membrane can select one among themselves using the tournament algorithm, and the parent can then forward the request to its parent in turn. The one membrane that succeeds to reach the skin level gains entry to the critical section. The number of rounds $k$ in this algorithm will be equal to the number of levels $l$ of the system. The pseudocode for the algorithm is mentioned below. The conventional algorithm is used by the term `tournament` and the list of membranes *list* is passed to it. The algorithm returns the *id* of the membrane that succeeds in the tournament algorithm.

**Algorithm: Tournament**

$l$ represents the maximum depth of the system,
$m_j$ is the parent membrane of $m_i$.

1. for $k = l$ downto 1
2. 1) all parent membranes $m_i$ at depth $k$:
3. if ($list \neq \phi$)
4. $w =$ tournament ($list$)
5. else
6. $w = -1$
7. end if
8. send $w$ to its parent, if any.
9. 2) all leaves $m_i$ at depth $k$:
10. if (need access to critical section)
11. $w = i$
12. else

```
13.              w = −1
14.          end if
15.          send w to its parent m_j, if any.
16.      3) all parent membranes m_i at depth k − 1
17.          for p = 0 to c − 1 (c is the number of children membranes)
18.              receive w_{i_p} from m_{i_p}
19.              if (w_{i_p} ≠ −1)
20.                  add w_{i_p} to list
21.          end for
22. end for
```

The first step is not executed in the algorithm when $k = l$. This is because there are no parent membranes at depth $l$ since that is the maximum depth of the tree. The above algorithm is good appropriate for P systems and it provides mutual exclusion with no deadlock and no lockout.

## 6 Fault Tolerant Consensus

For a system to coordinate effectively, often it is essential that every membrane within the system agree on a common course of action. With the help of leadership election, and a subsequent broadcast/flooding, it is possible for a consensus to be reached. This section discusses the consensus problem and fault tolerance (viz. reaching a consensus despite having failures within parts of the system).

### 6.1 The Consensus Problem

Consider a system in which each membrane $m_i$ needs to coordinate with the rest of the processors and choose a common course of action, i.e. agree upon a value for the variable *decision*. A solution to the consensus problem must guarantee the following:

- *Termination*: In every admissible execution, all the non-faulty nodes must eventually assign some value to *decision*.
- *Agreement*: In every admissible execution, all the non-faulty nodes must not decide on conflicting values.
- *Validity*: In every admissible execution, all non-faulty nodes must make the correct decision, i.e. must choose the correct value for *decision*. In other words, that if the consensus problem in question is choosing the maximum value from a certain set of numbers, then the *decision* must actually be the maximum value from the given set of input values.

Clearly the consensus problem is an important one, and the process would be disturbed in the presence of nodes which behave in an undesirable manner. However, within certain restrictions, it is possible to achieve a fault-tolerant consensus.

### 6.2 Failures

A failure is said to occur when a membrane behaves abnormally. There are two basic types of failures. *Simple failures* are when some membranes within the membrane system just stop functioning and do not ever recover, but wrong operations are never performed. The *Byzantine failures* are when some faulty membranes may behave in an unpredictable manner, contrary to a process which would help to reach a consensus.

**The Simple Failure Case**
The most important parameter which needs to be determined is $f$, the maximum number of membranes that can fail so that the consensus may still be achieved. Such a system is called an $f$-resilient system. We have the following results:

**Lemma 1.** *In every execution at the end of $f + 1$ rounds, all non-faulty membranes have the same set of values to base their decision upon.*

**Theorem 2.** *It takes an upper bound of $f + 1$ rounds to solve the consensus problem with simple failures in an f-resilient system.*

**Algorithm: Consensus**
Initially, every membrane $m_j$ has some value $x_j$ which it needs to send to all other membranes and ultimately reach a consensus based on these values.

In every round $k$, $(1 \leq k \leq f + 1)$, $m_i$ behaves as follows:

1. Send $x_i$ to all membranes within parent's membrane
2. Receive $x_j$ from $m_j$.
3. Add $x_j$ to an array (vector) $V$.
4. If $k = f + 1$ make decision based on the values stored in $V$.

**The Byzantine Failure Case**
This type of failure is more severe. The case is called the Byzantine failure because of a metaphorical description of a plan of action taken by several divisions of the Byzantine army (i.e. with traitors) to attack an enemy city [5]. In the Byzantine case, the faulty membranes behave arbitrarily and even maliciously. In a $f$-resilient Byzantine system there exists a subset of at most $f$ "Byzantine faulty" membranes. It becomes difficult to distinguish between a functional and a Byzantine faulty membrane, because unlike a membrane that crashes and simply stops sending objects, a Byzantine faulty membrane continues to send objects which may hamper the consensus process that requires membranes to agree on a common action based on their possible conflicting inputs. It is known that a consensus can be reached only if less than a third of the processors are Byzantine faulty processors [5]. The result is also true for membranes.

**Theorem 3.** *In a system with $n$ membranes, having $f$ Byzantine membrane, there is no algorithm to solve the consensus problem if $n < 3f$.*

**Theorem 4.** *In order to reach consensus in an $f$-resilient system, every non-faulty membrane must send at least $f + 1$ objects to all other non-faulty membranes to meet the requirements of the consensus problem.*

## 7 Example and Implementation

In this section we present an example describing an immune response system against virus attacks. This example is implemented using a membrane system library to emulate the main functions of a membrane system, and Message Passing Interface library that takes advantage of the highly parallel features provided by the model. Message Passing Interface (MPI) is still the most popular environment in the field of parallel computing, though many new parallel languages and tools were introduced.

When a virus enters a cell, it tries to destroy the host cell and all the surrounding cells by periodical replication and propagation. The human immune system is in charge of producing suitable antibodies which can counter-attack the virus. In the event that the antibody is not present, it needs to be transported (through intercellular communication) from the cell producing it to the place where it is required. The survival of the cell depends on the availability of these antibodies.

### 7.1 Terms

- Immune Response: The immune response is the way the body recognizes and defends itself against microorganisms, viruses, and substances recognized as foreign and potentially harmful to the body.
- Antibody: Antibodies are special proteins that are part of the body immune system. White blood cells produce antibodies to neutralize harmful germs called antigens.
- Virus: An infectious particle composed of a protein capsule and a nucleic acid core, which is dependent on a host organism for replication.
- Virus Propagation: The process by which the virus multiplies and sends copies of itself to the inner cells.
- Virus Neutralization: The process by which the virus is deactivated (destroyed) by the corresponding antibody.
- Clean cell: A cell which is free from any antigen.
- Infected cell: A cell which has at least one virus present in it.
- Virus Maturity Period: This is the time duration in which the virus is inactive, after which it regularly replicates.
- Virus Propagation Period: A mature virus regularly replicates after a certain number of time units, and this period is defined as the virus propagation period.

### 7.2 Problem Definition

Given an initial membrane structure with the viruses and antibodies, it is useful to know that when an equilibrium is reached, whether all the cells are *clean* or some cells remain *infected*. From a pharmaceutical perspective, this tells us whether or not the membrane requires any external medicinal supply (or whether it is strong enough to resist the virus). Certain configurations worsen at every

subsequent phases i.e. more viruses survive and antibodies slowly get depleted. The detection of such patterns in early stages would increase the chances of cleaning the cells.

### 7.3 P System Perspective

The organism which the virus infects is represented as a skin membrane in the P system. The cellular hierarchy of the organism is modelled as the tree structure of the membranes. The virus and antibodies are the objects in the system. The virus and antibody properties (e.g. the type and life) are the symbols of the objects. We use the following rules:

1. The skin membrane has unlimited supply of antibodies of all types.
2. An antibody of type $a_i$ is required to neutralize the virus $v_i$.
3. For virus propagation:
   - each virus has a maturity period, after which it can reproduce,
   - thereafter, it reproduces regularly after a fixed propagation period,
   - the virus child thus created may be sent to any one of the children membranes in a random manner.
4. A membrane requiring an antibody sends a request to its parent for that particular antibody.

For this problem, a distributed computing approach is given by the following algorithms:

- Leadership Election: Identifying whether a cell is clean or infected is analogous to whether the virus wins or not in an leader election algorithm;
- Synchronization: Communication between membranes to maintain the required balance of antibodies in order to reach a clean state.

### 7.4 Equilibrium State Determination

Determining whether the virus will survive or not is not a trivial problem. The sufficient condition for the equilibrium state can be written as $M > 2D$, where $M$ is the maturity period of the virus, and $D$ is the maximum distance between a virus and the skin. This is easy to see as it will take exactly $2 \times D$ time for a membrane to receive the antibody after it has requested for it. If the above condition is satisfied, the virus will be destroyed before it reproduces. This condition has to be true for all the membranes. However, the necessary condition is more complicated. Since, some antibodies may be present in earlier stages as well (e.g. one of the membranes other than the skin membrane) the membrane can receive the antibody earlier. Thus, in practice a precise analysis is required. Moreover, each child virus will have its own life-cycle of maturing and reproducing simultaneously, making the mathematical formulation rather complicated.

The following table shows the number $n$ of viruses at different times $t$. In this example $M = 3$ and the virus propagation period $P = 1$.

| Time | Total | New | Mature |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 2 | 1 | 0 | 0 |
| 3 | 2 | 1 | 1 |
| 4 | 3 | 1 | 1 |
| 5 | 4 | 1 | 1 |
| 6 | 6 | 2 | 2 |
| 7 | 9 | 3 | 3 |
| 8 | 13 | 4 | 4 |
| 9 | 19 | 6 | 6 |
| 10 | 28 | 9 | 9 |

From the table we can observe that $n(t) = n(t-1) + n(t-3)$; generalizing, we get $n(t) = n(t-P) + n(t-M)$. It is possible to express $n(t)$ as a polynomial function of $t$. The coefficients of the polynomial expression are provided by the the roots of the equation $t^M - t^{M-P} - 1 = 0$. It is difficult to solve this equation manually; a recursive algorithm can easily be implemented at the start of the simulation to decide the state.

### 7.5 Algorithm: Immune Response

In each round we have

1. An exchange of antibodies:
   (a) every parent sends its antibodies to its children as requested in the previous round;
   (b) every child receives antibodies as sent by parent;
2. a virus propagation:
   (a) increment virus life and check for reproduction;
   (b) send virus to children if required;
   (c) receive virus from parent (if not skin).
3. **Compute leader**: for each round, check if virus dominates or is destroyed.
4. **Synchronization**: send antibody requests to all parents.

### 7.6 Implementation

For implementing this example, we have used the Message Passing Interface (MPI). MPI is a standard developed to enable portable message passing applications, and though many new parallel languages and environments are introduced, it is still very popular in the field of parallel computing. MPI is a library of functions and macros that can be used in the Fortran, C, C++ and Java programs. MPI programming means that you write your program in C, C++ or Java, and when the time comes for parallel processes to communication or synchronize, you should explicitly call the MPI send or receive function to

help. MPI send function sends a message to the named target process, while in the target process, a correspondent receive function must be set to make the corresponding work. MPI is quite easy to use. You need to master around six commands to write simple programs; they are MPI_Init(), MPI_Comm_rank(), MPI_Comm_size(), MPI_Send(), MPI_Recv() and MPI_Finalize(). To use the MPI system and functions, the header file mpi.h should be included. Different processes are identified with their task ID's; the MPI system assigns each process a unique integer called as its rank (beginning with 0). The rank is used to identify a process and communicate with it. Each process is a member of a communicator; a communicator can be thought of as a group of processes that may exchange messages with each other. By default, every process is a member of a generic communicator environment (it could be interpreted as the skin in a membrane system). Although we can create new communicators, this leads to an unnecessary increase in complexity. The processes can be essentially identical, i.e. there is no inherent master-slave relationship between them. So it is up to us to decide who will be the master and who will be the slaves. A master process can distribute data among the slaves. Once the data is distributed among the slaves, the master must wait for the slaves to send the results and then collect their messages. Packing and decoding is handled by MPI internally. The code for the master as well as the slaves could be in the same executable file. More details can be found in [6,8]. Our implementation is written in C and uses the parallel environment provided by the MPI library.

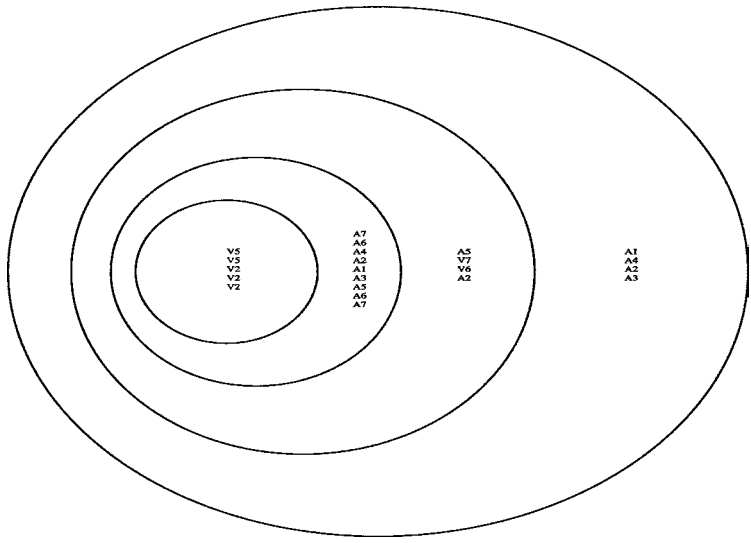



**Fig. 1.** Example of an output diagram

The implementation steps were:

1. A P system library was written to emulate the functions of a P system. We assume that each membrane is a processor.
2. An MPI program was written for the simulation of the various rounds of the system, implementing the algorithm presented above.
3. An interface was written to create an additional level of abstraction between the library and the MPI program, in order to hide the implementation details of the P system library.
4. An automated graphical output is generated at the end of the simulation making use of XFig and LaTeX. The output at the end of each round is used to encode an XFig diagram, and these are included in a LaTeXpresentation.

## References

1. H. Attiya, J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*, McGraw-Hill, 1998.
2. G. Ciobanu. Distributed algorithms over communicating membrane systems, *BioSystems*, Elsevier, to appear.
3. G. Ciobanu, D. Paraschiv. P System Software Simulator, *Fundamenta Informaticae* vol.49 (1-3), 61-66, 2002.
4. G. Ciobanu, D. Dumitriu, D. Huzum, G. Moruz, B. Tanasă. Client-Server P Systems in modeling molecular interaction. In Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): *Membrane Computing 2002*, Lecture Notes in Computer Science - this volume, Springer, 2002.
5. L. Lamport, R. Shostak, M. Pease. The Byzantine generals problems. *ACM Trans. Program. Lang. Syst.* vol.4(3), 382-401, 1982.
6. P. Pacheco. *Parallel Programming with MPI*, Morgan Kaufmann Publishers, 1997.
7. Gh. Păun. Computing with membranes, *Journal of Computer and System Sciences*, vol.61, 108-143, 2000.
8. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra. *MPI–The Complete Reference vol.1, The MPI Core*, 2nd edition, MIT Press, 1998.

# Client–Server P Systems in Modeling Molecular Interaction

Gabriel Ciobanu[1], Daniel Dumitriu[2], Dorin Huzum[2], Gabriel Moruz[2], and Bogdan Tanasă[3]

[1] National University of Singapore, School of Computing
gabriel@comp.nus.edu.sg
[2] "A.I.Cuza" University of Iaşi, Romania
{danield,huzzy,gabbym}@infoiasi.ro
[3] Institute of Computer Science, Romanian Academy, Iaşi
bogdan@iit.tuiasi.ro

**Abstract.** We present a new version of P systems called *Client–Server P Systems* (CSPS). The client membranes are characterized by their states; the server membrane stores the states of the clients and triggers the corresponding interaction rules. We show that CSPS have the same expressive power as Turing machines. CSPS is used to model various molecular processes in which interaction and state transitions are causally linked. Signaling pathways and T cell activation are described by using a CSPS software environment called MOlNET (MOlecular NETworks). MOlNET can describe the dynamics of molecular interactions, including both qualitative and quantitative aspects and simulating the signaling pathways that tune the activation thresholds for T cells.

## 1 Introduction

Membrane computing is based on *membrane systems* or P systems, a new class of distributed and parallel computing devices introduced in [12]. The approach is based on hierarchical systems: finite cell-structures consisting of cell-membranes embedded in a main membrane called the *skin*. The membranes determine *regions* where *objects*, elements of a finite set, and evolution rules can be placed. The objects evolve according to given *rules* associated with a region. Objects may also move between regions. A *computation* starts from an initial configuration of the system, and terminates when no further rule can be applied. A software simulator of membrane systems is presented in [3].

Hierarchical systems are well-known structures in computer science, and the notion of computation based on evolution rules is common. The interpretation of the computation is rather new: the result of a computation is a multiset of objects collected in the output cell or sent out of the system. The behaviour of the whole system is obtained by combining the resulting multisets, or considering the multiplicity of objects present in a specific output membrane of a final configuration.

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 203–218, 2003.

According to many authors, P systems are not created to model biological systems, although similarities can be observed. Since their introduction, many results of universality were proved and several theory problems could be explained in an easier and elegant manner, with the help of formal languages. Yet, the subject is far from being exhausted; new properties are discovered, as well as connections with already known concepts. There is still a need to find more connections with the applied computer science and approaches that could prove themselves useful in practice as well as in theory. We present a new version of P systems, one related to the applied computer science and molecular biology. Trying to strengthen the connection between P systems and biological systems, we introduce P systems of interaction, considering the client–server model used for process communication in computer networks.

We use a new version of P systems called Client-Server P Systems (CSPS) to model molecular processes as signaling pathways and T cell activation. A living cell is a complex system where the interactions among its components provide structure, mediate reactions, and perform functions. Different cell processes (such as T cell activation) are triggered mainly by protein-protein interactions. Specific and productive interactions change consequently the states of the interacting proteins (such as in signal transduction). Such molecular processes are modeled in MOlNET, a CSPS software environment more meaningful for biologists and for their goals.

The paper is organized as follows. Section 2 presents the client-server P systems, showing that they have the same computational power as Turing machines. Section 3 briefly describes the important notion of a signaling pathway and T cell activation. Section 4 presents MOlNET as a CSPS software environment. MOlNET is based on the client–server model of the computer networks. After a brief presentation of MOlNET, emphasizing on its components and their behaviour similar to CSPS, we simulate the signaling pathways that finely tune the T cell activation thresholds taking into consideration both qualitative and quantitative aspects.

## 2 Client–Server P Systems

A formal description of a *P system* can be found in [12]. It is basically composed of a *membrane structure*, consisting of several membranes that do not intersect, and a *skin membrane*, surrounding them all; outside the skin membrane lies the *environment*. The membranes delimit regions, numbered accordingly, and contain originally multisets of *objects*, as well as *evolution rules* involving objects (and possibly priorities for rules).

This is the initial state of the P system. In each step, rules are applied nondeterministically and in a maximal and parallel manner, in all membranes – in other words, rules and objects are randomly chosen, all objects that can be involved really are, and all chosen rules are applied in parallel.

The objects can pass membranes, even to or from the outside of the system; in this way we obtain *transitions* between *configurations* of the system. A sequence

of transitions constitutes a *computation*; this *halts* if at one moment, no rule can be applied anymore in the system. When getting a halting computation, we collect its *result* by counting the objects that ended in the *output membrane*.

These are the general P systems, and many other variants and classes were introduced starting from this simple description. For our paper, a particular kind of P systems is of special interest, *P systems with symport/antiport rules*; they take benefit from the communication that is intensively used during computations. The specificity of this type of P systems lies in the form of their rules, which can be one of:

– $(a, in), (a, out)$: object $a$ can pass through a membrane by itself (*uniport rules*),

– $(ab, in), (ab, out)$: objects $a$ and $b$ can pass through a membrane only together, in the same direction (*symport rules*), and

– $(a, out; b, in)$: objects $a$ and $b$ can pass through a membrane only together, but in different directions (*antiport rules*).

There is a number of results on this type of P systems [11]; generally, they take into consideration the number of membranes and the *weight* of the port (i.e. the number of objects involved in an antiport or symport rule). A relation of inverse proportionality between these two parameters could be observed – that is, in order to obtain computational universality, one has to increase the weight of the ports when decreasing the number of membranes, and reversely. In the proofs from this paper, we use some results and techniques rooting in the formal language theory that can be found in [14], for example.

A *matrix grammar with appearance checking* is defined as a construct $G = (N, T, S, M, F)$, where $N, T$ are disjoint alphabets, $S \in N$, $M$ is a finite set of sequences of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, (called *matrices*) of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$), and $F$ is a set of occurrences of rules in $M$.

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there exists a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in $M$ and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w'_i A_i w''_i, w_{i+1} = w'_i x_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$, or $w_i = w_{i+1}$, $A_i$ doesn't appear in $w_i$, and the rule $A_i \to x_i$ appears in $F$. (The rules of a matrix are applied in order, possibly skipping the rules in $F$ if they cannot be applied, hence the grammar's name.)

The language generated by $G$ is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by $MAT_{ac}$. It is known that matrix grammars with appearance checking generate the family $RE$ of recursively enumerable languages. A matrix grammar $G$ with the notations made above is in the *binary normal form* if $N = N_1 \cup N_2 \cup \{S, \#\}$, all sets disjoint, and the matrices in $M$ are in one of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$,
2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$,
3. $(X \to Y, A \to \#)$, with $X, Y \in N_1, A \in N_2$,
4. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2, x \in T^*, |x| \leq 2$.

Moreover, there is only one matrix of type 1 and $F$ consists exactly of all rules $A \to \#$ appearing in matrices of type 2; # is a trap-symbol (it is never removed once introduced). A matrix of type 4 is used only once, in the last step of a derivation. According to Lemma 1.3.7 of [6], for each matrix grammar, there exists an equivalent matrix in the binary normal form.

### 2.1 Description

We formalize a client–server model complying with the following description. The clients are characterized by their states, whereas the server stores the current states of clients and also interaction rules defined over these states. When two clients can interact, the server notifies them, supplying at the same time the corresponding rule. The clients interact and send their new states to the server, thus making the model consistent.

A *Client–Server P System* (CSPS) is a P system composed of elementary membranes (except for the skin), state-objects for all of them minus one (the server) and objects modelling the real communication channels between the clients. All rules are of symport type. A CSPS resembles the original client–server model by working with information and not using rules with creation/destruction of objects, but merely the power of communication. In a formal notation, the CSPS contains the skin membrane (numbered 1), $m$ membranes each representing a client (numbered from 2 to $m+1$) and one for the server, numbered $(m+2)$, all arranged inside the skin membrane. We have state-objects for possible states of clients and rule-objects.

A rule-object $\eta_{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}$ has the following meaning: two clients defined by states $\alpha_1$ and $\alpha_3$ can interact and pass to states $\alpha_2$ and $\alpha_4$, respectively; at the same time, it is possible to obtain a supplementary information, $\alpha_5$.
Let $\Pi$ be our CSPS:

$$\Pi = (V, \mu, w_1, \ldots, w_{m+2}, w_e, M_e, R_1, \ldots, R_{m+2}, m+2)$$

where:

1. $V = A \cup B$, with $A, B$ disjoint sets constructed as follows:
   – $A = \bigcup_{i=2}^{m+1} S_i$, where $S_i$ is the set of states for client $i$;
   – $B = \{\eta_{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5} \mid \alpha_5 \in M_e \cup \{\lambda\}, \alpha_i \in A \cup M_e,\ 1 \le i \le 4$, where $\alpha_1 + \alpha_2 \Rightarrow \alpha_3 + \alpha_4 + \alpha_5$ is an interaction rule$\}$,
2. $\mu = [_1 [_2 ]_2 \ldots [_{m+2} ]_{m+2} ]_1$,
3. $w_1 = \emptyset$,
   $w_{m+2} = B \cup S_{initial}$, where $S_{initial} = \{s_2, s_3, \ldots, s_{m+1}\}$, $s_i \in S_i$, $2 \le i \le m+1$ (the initial state of each client),
   $w_i = S_i \setminus \{s_i\}$, $s_i \in S_{initial}$, $2 \le i \le m+1$,
   $M_e = A$ (multiset),
4. $R_1 = \{(\alpha_j\ \alpha_k\ \eta_{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}, out) \mid j \in \{3,4\}, k \in \{1,2\}, j-k \ne 2, \alpha_j, \alpha_{j+2} \in A, \alpha_k, \alpha_{k+2}, \alpha_5 \in M_e i\}, \cup \{(\alpha_3\ \alpha_4\ \eta_{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}, out) \mid \alpha_i \in A,\ 1 \le i \le 4,\ \alpha_5 \in M_e\} \cup \{(\alpha_3\alpha_4\alpha_5\eta_{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}, in) \mid \alpha_i \in A \cup M_e,\ 1 \le i \le 5\}$
   $R_{m+2} = \{(\alpha_1\ \alpha_2\ \eta_{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}, out),\ (\alpha_3\ \alpha_4\ \alpha_5\ \eta_{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}, in) \mid \alpha_i \in A \cup M_e,\ 1 \le i \le 4,\ \alpha_5 \in M_e \cup \{\lambda\}\}$,

$R_i = \{(\alpha_j\ \eta_{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}, in),\ (\alpha_{j+2}\ \eta_{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}, out) \mid j \in \{1,2\},\ \alpha_j, \alpha_{j+2} \in S_i\},\ 2 \leq i \leq m+1$.

Inside the server membrane there are state-objects (representing the current states of the clients) and rule-objects. When two state-objects can be combined according to a rule given by a rule-object, the server membrane gives a "signal" to the respective client-membranes.

The meaning of the rule $(\alpha_1\ \alpha_2\ \eta_{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}, out) \in R_{m+2}$ is the following: the clients represented by membranes $i$ and $p$ (where $\alpha_1 \in S_i$ and $\alpha_2 \in S_p$) can interact according to the rule described by the rule-object $\eta$; as a result, these three objects (the current states and the rule-object) exit the server region. At this moment the involved client-membranes absorb their own state-objects and the rule-object $\eta$ (by means of rules $(\alpha_1\ \eta_{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}, in) \in R_i$ or $(\alpha_2\ \eta_{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}, in) \in R_i$, and similarly for membrane $p$). Then they release for further use their new states and the rule-object into the skin membrane, by $(\alpha_3\ \eta_{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}, out) \in R_i$ or $(\alpha_4\ \eta_{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}, out) \in R_i$ (and similarly for $p$). If $\alpha_5 \neq \lambda$, the supplementary information is brought in from the environment with rules of $R_1$.
We emphasize the fact that the notifications of clients and the interactions between them take place in a parallel manner.

### 2.2 The Computational Power

We show that the Client–Server P Systems are computationally universal, i.e. they have the same computational power as Turing machines. Because of the characteristics of the model, it is more convenient to use in the proof a slightly modified grammar. Given a general matrix grammar $G = (N, T, S, M, F)$ in the binary normal form, let us denote by $G' = (N', T, S, M', F')$ the grammar constructed as follows:

- $N' = N_1 \cup N_2 \cup N_3 \cup \{S, \#\}$, all sets disjoint, with $N_3 = \{u' \mid u \in N_2\}$; we define the bijection $corresponding : N_2 \to N_3, corresponding(u) = u'$;
- the rule $(S \to XA) \in M$ is replaced in $M'$ by $(S \to XA')$, with $X \in N_1, A' \in N_3, A' = corresponding(A)$ .
- we keep the matrices of $G$ of types 2, 3, 4 and:
  - for each matrix of type 2, $(X \to Y, A \to x) \in G$, we add $(X \to Y, A' \to x_1)$ with $corresponding(A) = A' \in N_3, x_1 \in (N_3 \cup N_2 \cup T)^*$; if $1 : x \in N_2$ then $1 : x_1 = corresponding(1 : x)$ and $2 : x_1 = 2 : x$, else $x_1 = x$;
  - for each matrix of type 3 and 4, we add the corresponding rule: $(X \to Y, A' \to \#)$ or $(X \to \lambda, A' \to x)$, with $X, Y \in N_1, corresponding(A) = A' \in N_3, x \in T^*, |x| \leq 2$.
- $F'$ consists of rules in $F$ plus added rules of type $A' \to \#$, where $A' \in N_3$, $A' = corresponding(A)$.

In this definition we use the notation $\alpha : x$ representing the $\alpha^{th}$ element of the sequence $x$ (for instance, $2 : x$ means the second element of $x$).

We prove that the two grammars are equivalent, so we can further use any of them, depending on which of them fits better the goal.

**Lemma 1.** $G \equiv G'$.

*Proof.* We show that every derivation of $G$ can be simulated in $G'$ and vice versa. This is proved by induction on the length of derivation. □

**Notations:**
Given a system $\Pi$ as defined in the previous section, the set of all numbers computed by $\Pi$ is denoted by $N(\Pi)$. We denote by $NCSP_{m,p}$ the family of all sets $N(\Pi)$ computed by CSPS of degree at most $m \geq 1$, using symport rules of weight at most $p \geq 1$. The weight of a symport rule $(a_1 \dots a_n, in/out)$ represents the number $n$ of the objects appearing in the rule. For instance the weight of $(a\,b\,c\,, in)$ is 3. We denote by $N'RE$ the family of all recursively enumerable sets of non-null numbers. We prove that for $m = p = 4$, CSPS are computationally universal (except that they cannot yield the number 0).

**Theorem 1.** $NCSP_{4,4} = N'RE$.

*Proof.* The direct inclusion can be proved in a straightforward manner.

For the reverse one, we use a method often employed when proving results about universality of different types of P systems, namely exploiting the properties of the matrix grammars with appearance checking. Yet, because of model's characteristics, it is more convenient to use in the proof a matrix grammar modified as above, that is having the set of non-terminals differently partitioned and the rules modified accordingly.

It is known that matrix grammars with appearance checking generate the family $RE$ of recursively enumerable languages. Taking this into consideration, as well as the results regarding the equivalence of the general matrix grammars and the ones in the binary normal form, and the latter with our modified type (see Lemma 1), we construct a client–server P system of degree 4 and symport weight 4 which simulates the derivations of the "modified" grammar. Let this be $G' = (N', T, S, M', F')$, with $N = N_1 \cup N_2 \cup N_3$. For every matrix $(X \to Y, A \to \alpha_1\alpha_2)$ in $G'$, with $X, Y \in N_1$, $A \in N_2 \cup N_3$ and $\alpha_1, \alpha_2 \in N_2 \cup N_3 \cup \{a, \lambda\}$, we have an associated object of type $\eta$: $\eta_{XAY\alpha_1\alpha_2}$. For the rules $(X \to \lambda, A \to \alpha_1\alpha_2)$ and $(X \to Y, A \to \#)$ we introduce the objects $\eta_{XAt\alpha_1\alpha_2}$ and $\eta_{XAYb}$, respectively. In order to be able to eliminate these objects $\eta$ from the output membrane (region) at the end of the derivation, we introduce some special objects $s$ and $s'$ for every object $\eta$.

A sketch for the simulation of a general derivation for the case $A \in N_3$ can be seen in Figure 1. At the beginning, in membrane 2 there are the elements of $N_1$, in membrane 3 there are the elements of $N_3$ and the terminal $a$, while in 4 we have the right hand side of the starting rule of grammar and the rule-objects. In the environment there are the elements of $N_2$ and the terminal $a$, in an infinite number of copies. In the case when $A \in N_2$, the system functions in a similar manner, except for the fact that steps 4 and 5 are skipped, and in step 6 the rule applied is $(Y A\, \eta_{XAY\alpha_1\alpha_2}, out) \in R_1$.

We use a matrix $(X \to \lambda, A \to x)$ of type 4 in $G$ only when an object $t$ appears in membrane 4, triggering the mechanism for removing the objects

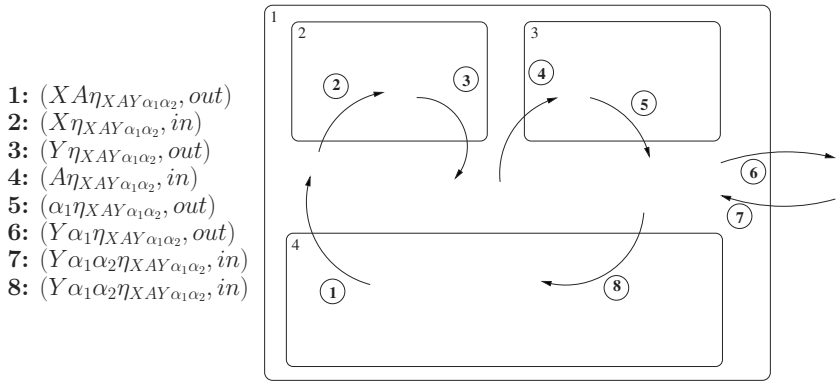



**Fig. 1.** A client-server P system of degree 4 and symport weight 4

$\eta$ from the output membrane. When a matrix $(X \to Y, A \to \#)$ is used in $G'$, the trap-object $b$ is introduced in membrane 4 preventing the end of the computation. □

Having introduced the CSPS as above, one question arises: are there problems that could be more naturally described (and *solved*) by CSPS than by other P systems? A positive answer comes from biology. We have detected that CSPS approach has a very interesting biological feedback when it is used to describe the T cell activation.

## 3 Signaling Pathways and T Cell Activation

**Signaling pathways**
Many biological activities depend on the ability of proteins to communicate specifically with each other and with other molecules. This is particularly obvious for the signaling pathways that operate in living cells. These signaling pathways allow the cell to receive, process and respond to various *signals* from its environment. The signals are molecules (hormones, neurotransmitters, cytokines) that indicate the beginning and the termination of one or more intracellular processes. The *signal transduction* refers to the process by which the signals are transmitted via receptors to the interior of the cell through the signaling pathways. Classically, a linear signal transduction pathway can be described as follows. A first messenger molecule (the signal) outside the cell is sensed at the cell membrane by a receptor. Receptor binding is transduced into an intracellular event by activation of enzymes (PLC, for instance) that synthesizes second messenger molecules (e.g. DAG, IP3). These ones promote the covalent modifications (by protein kinase or protein phosphatase activities) and allosteric regulation of other intracellular proteins that ultimately cause specific changes in gene expression. In living cells, these linear signaling pathways cross-talk each other and form a network of interactions. This signaling network has several

emergent properties that the individual pathways do not have [1]. We consider the signaling network that grounds T cell activation as a case study of our model. When a T cell recognizes a foreign antigen, it initiates several signaling pathways and the cell activates. The recognition of foreign antigen is an extremely *sensitive*, *specific* and *reliable* process and models for T cell signaling are necessary to understand how these properties arise. So far, the study of T cell activation has benefited from the use of mathematical models [2] or are based on Boolean formalisms [8].

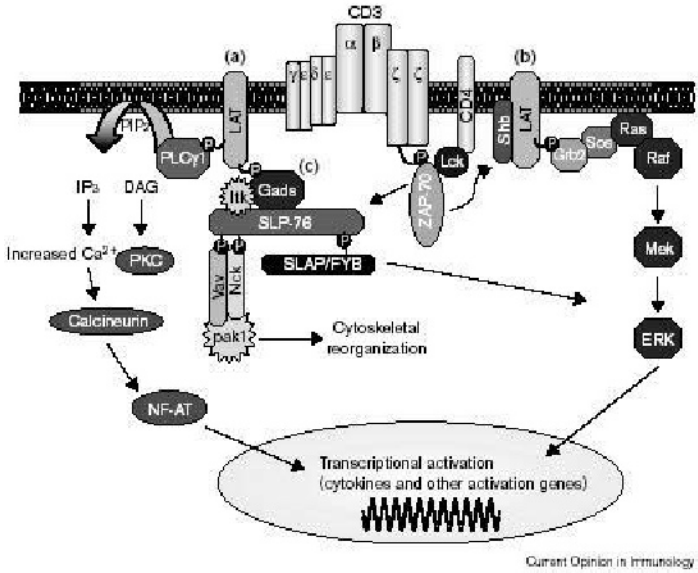



**Fig. 2.** TCR signaling pathways

## T Cell Activation

T cells play an important role in orchestrating the immune responses to foreign aggressors. The key event for T cell activation is an appropriate interaction between the T cells armed with T cell antigen receptors (TCR) and the professional antigen presenting cells (APC). TCR only recognizes the foreign antigen in the form of short peptides presented in the groove of a molecule on the surface of the APC known as the major histocompatibility complex (MHC). The physical interaction of TCR with MHC-peptide complexes is unique among signaling systems in that it takes place over a continuum of binding values. The recognition of antigen initiates signal transduction. This can be broken down into series of discrete steps that are related to molecular events (interactions, state transitions) within the signaling pathways. These are shown in Figure 2, reprinted from [10]. The integration of distinct branches of the TCR-induced signaling pathways (Ras/MAP kinase pathway, $Ca^{+2}$/calcineurin pathways) results in activation

of distinct transcriptional factors that coordinate and regulate gene expression and cell activation. Other signals (costimulatory signals) also function in T cell activation by amplifying TCR signaling. The signaling pathways are modeled in MOlNET and the network behaviour is traced out both qualitatively and quantitatively.

## 4 MOlNET: A CSPS Software Environment

In this section we present MOlNET, a software environment for CSPS, first introduced in [4]. The link between CSPS and MOlNET is given by the fact that each component of MOlNET has a clear correspondent in CSPS, with the same role and behaviour. We use MOlNET to simulate the signaling pathways that tune the activation thresholds for T cells, providing insights on both quantitative and qualitative aspects.

### 4.1 Description

In every biological interaction, one or both interacting molecules undergo a transition to a new state. Consequently, a MOlNET structure contains information about each of the interacting molecules (name, type, location inside cell and number of relevant domains with respect to a given interaction, if the molecule is a protein). For each protein it is possible to mention the names of its domains and domain states (active/inactive).

Our system allows a user to define molecules (proteins, ions, lipids, DNA) that take part in a simulation, and also the interaction rules between them: protein-protein interactions, protein-DNA interactions and other features of molecular systems as a set of reaction rules. The user can observe, through various windows, dynamic changes in concentration of proteins and other chemical compounds of the cells. Using this software, we can tune up a specific behaviour and we can emphasize the problems related to interaction anomalies.

Our implementation choices were to use the C-standard programming language and the BSD-socket interface approach. The graphic server is written in Gtk 2.0 environment. The main components of our software system as well as the relationships established between them are shown in Figure 3.

The MOlNET architecture follows the definition of CSPS. Hereinafter we describe the components of our software environment, as well as the links with CSPS. The data server manages information about the interactions among the clients. It corresponds to the server membrane in CSPS, having the same role and behaviour. This component uses a non-directed graph defined by the molecules existing in the cell and their productive interactions. By productive interaction we understand that their interaction brings either qualitative or quantitative changes to the cell (the modification of a molecule state or concentration, forming or unbinding of a molecular complex).

The server carries out the decision function with regard to the possible molecule interactions. When it receives a request from a client, it checks the

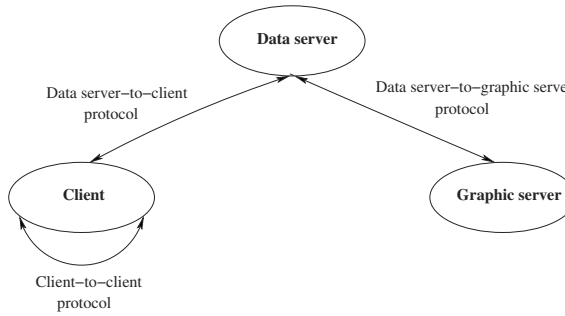



**Fig. 3.** Simplified MOlNET architecture

request type (new client or client modification), updates the graph and sends back the appropriate responses (a list of possible partners). It also keeps the amounts of each molecule existing in the cell.

A client represents a type of molecule; it corresponds to a client membrane in CSPS, having the same functionality. Each client saves information about its state: chemical structure, position, identification within computer networks (host+port). According to the model described above, a client (molecule) changes its position inside the cell and checks whether its interaction partners are available and close enough for an interaction. As soon as it participated to an interaction, a client notifies the server. The communication mechanism closely follows the CSPS formalism presented in Section 2. Specifically, the rule-objects of CSPS are interaction rules in MOlNET, initially stored in the data server. As a result of the message exchange, the data server and clients update their states, in the same way as transitions take place in CSPS.

The graphic server is an important part of the system, as it provides a friendly visual interface to the user. The users just introduce the input data, and then observe the behaviour of the system. A snapshot of the MOlNET user interface is given in Figure 4. In order to follow the real interactions between molecules, the graphic server offers the possibility for both defining the molecules that are in the system, and interaction rules between these molecules. If experiments prove that some rules were incorrect or if on the contrary they discover new interactions, it is easy to alter old rules or to add new ones, by the simple act of introducing new molecules and/or rules. We considered the signaling pathways that tune the T cell activation thresholds. The work is presented in the next section. The results obtained by running the system are represented as charts by the graphic server. The feature that allows direct interventions from outside is also carried out by this component.

As each particular client/server model should rely on its own protocol, we developed a set of protocols allowing the communication between data server and clients, data server and graphic server, clients themselves.

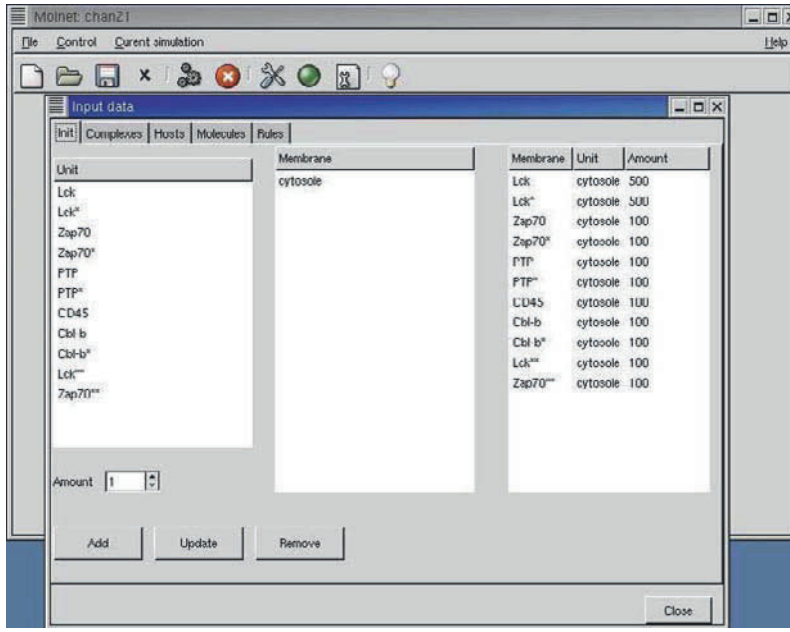



**Fig. 4.** A MOlNET screen for tuning the activation thresholds

### 4.2 Tuning the T Cell Activation Thresholds

MOlNET is able to catch both qualitative and quantitative aspects. The **qualitative approach** includes *connectivity data* (molecular interaction map) and *semantic data* (the effect of an interaction, mainly activation or inhibition). It allows a *dynamic description* of the signaling network. The **quantitative approach** includes kinetic data (amounts of molecules, reaction rates) and allows *predictions* on the network behaviour. A common difficulty when dealing with the simulation of molecular interactions is the lack of quantitative data.

T cell activation is a threshold phenomenon that is dynamically modulated (or tuned) during cell maturation [7]. It reflects the signal intensity that is necessary to increase the expression of specific genes (e.g. IL-2 gene). Both the emergence of threshold and its tuning depend on dynamic interplay between positive and negative factors. As T cells receive many signals from self antigens, they have to adapt their activation thresholds in such a way that self-stimuli fall under the threshold and consequently no response is elicited against self. Furthermore, non-self antigens provide stronger signals that overcome the activation threshold, the cell activates and an immune response is yielded. In our work, the activation threshold concept is considered on a molecular basis.

Although many receptor interactions may contribute positively or negatively to the setting of activation threshold, TCR signaling plays the dominant role and has its own signature. Zap70 activation and phosphorylation of LAT are hallmarks of TCR engagement and are essential for connecting to the major in-

tracellular signaling pathways ($Ca^{+2}$/calcineurin and Ras/MAPK kinases) that lead to T cell activation.

In the following, we investigate the role of Cbl-b in tuning the activation thresholds. Basically, we look for the influences that Cbl-b exerts on the level of activated Zap70. First, the model proposed in [2] is considered. Then, Cbl-b is added and the levels of Zap70* are measured. High levels of Zap70* may trigger cell activation, while levels below the threshold have not this effect. The expression levels of various signaling proteins vary during immune cell maturation (e.g the level of Lck declines during development whereas the level of SHP phosphatase increases); our work considers the heterogeneity of activation thresholds at the level of population of T cells (or T cell clones) rather than during the development of a single clone.

According to [2], the state of TCR signaling (hence the activation threshold) appears to be governed by a dynamic balance of kinases (Lck, Zap70) versus phosphatases (SHP1, SHP2). Moreover, there are some feedback interactions among Lck, Zap70 and phosphatases (the later are represented by a single variable called PTP). As shown in Figure 5, Lck* phosphorylates and activates both the ZAP70 (step 3) and the PTP (step 5). Zap70* activates Lck in step 1 (positive feedback), but phosphatase inactivates both Lck* (step 2) and ZAP70* in step 4 (negative feedback). The active state of an enzyme is denoted by "*".

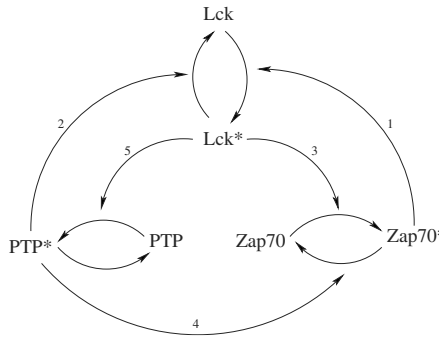



**Fig. 5.** Feedback interactions as TCR signals

These feedback combinations may have a nonlinear effect that alters the appropriate threshold for cell activation. Moreover, these interactions due to feedbacks provide robust mechanisms for antigen discrimination (self vs non-self). Taking into account the above particularities of the reaction rules, one can observe in Figure 6 the changes of Lck*/Lck-total and Zap70*/Zap70-total ratios after TCR engagement. For various amounts of Lck and Lck*, Zap70*/Zap70-total ratio has slight variations. If the cell activation threshold (namely its requirement for Zap70*) is below the Zap70* signal intensity, the cell is activated; otherwise (that is, if its threshold is above the actual signal), the cell is not activated. The cell response seems to be also sensitive to the variations of Lck and Lck*.

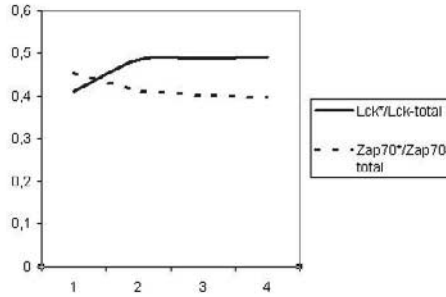



**Fig. 6.** *Lck*/Lck-total and Zap70*/Zap70-total levels after TCR triggering*

During experiments (represented by numbers 1 to 4 along the horizontal coordinate in Figure 6), the input values of Lck and Lck* varied between 10,000; 100,000; 500,000 and 1,000,000 molecules, while the input values of Zap70 and Zap70* were kept constant: 100,000. The kinetic constants were $k_1$=0,001, while $k_2$=$k_3$=$k_4$=$k_5$=1, where $k_i$ stands for the reaction $i$ of Figure 5.
Lck-total=Lck+Lck* and Zap70-total=Zap70+Zap70*.

Recent reports highlight that Cbl-b is a key regulator of activation thresholds in T cells. Many proteins associate with Cbl-b, including Lck* and Zap70*. Cbl-b mediates chemical modification (ubiquitination) of these activated kinases that targets them for degradation [13] (reactions 8 and 9 in Figure 7). Degra-

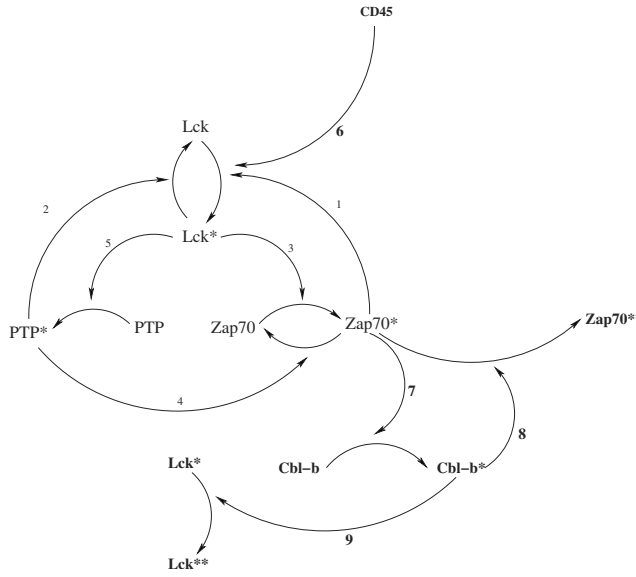



**Fig. 7.** *Cbl-b alters the signaling pathways* The specific chemical modifications (due to ubiquitination) that target the active enzymes to degradation are denoted by "**".

dation of active kinases results in the reduction of the activation of downstream signaling proteins. Furthermore, degradation of Lck can reduce the activation of Zap70, as shown in Figure 7. These events raise the threshold requirements for cell activation and prevents the development of autoimmunity [15]. Moreover, following TCR ligation, Zap70* activates Cbl [10] (reaction 7). Additionally, CD45 activates Lck (reaction 6).

All these molecular events finely tune the signal intensity in such a way that it draws nearer to or deviate from the activation threshold. The changes in the Lck*/Lck-total and Zap70*/Zap70-total ratios are shown in Figure 8. When the amounts of Cbl-b (and Cbl-b*) vary and amounts of Lck (and Lck*) vary as well, Zap70*/Zap70-total ratio still has slight variations (as in Figure 6). But when the amounts of Cbl-b and Cbl-b* equal 500.000 molecules, for an activation threshold set around 0.45 (that is Zap70*/Zap70-total=0.45, a thin red line in our picture), the cell could either activate or not (during experiment 2, the signal intensity is below the threshold, while in the experiment 3 the threshold is overcome). These outcomes are produced by differentially regulating the amount of Lck* within the cell. In other words, Cbl-b fine tunes T cell reactivity, and this also depends on the amount of Lck*.

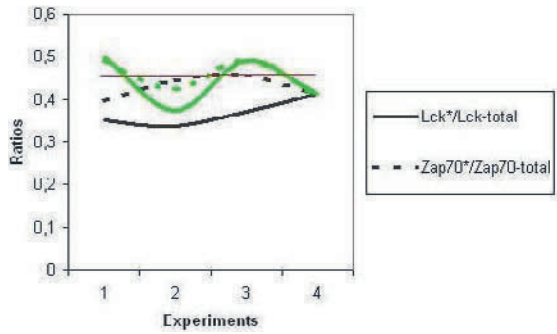



**Fig. 8.** *Lck*/Lck-total and Zap70*/Zap70-total levels after TCR triggering and Cbl-b activation*

During our software experiments (represented by numbers 1 to 4 along the horizontal coordinate in Figure 8), the input values of Lck and Lck* varied between 10,000; 100,000; 500,000 and 1,000,000 molecules, while the input values of Zap70 and Zap70* were kept constant: 100.000. Cbl-b and Cbl-b* were set to 10,000 (dark lines), and then they were set to 500,000 (green lines). The kinetic constants associated to the corresponding reactions of Figure 7 were $k_1$=0,001, $k_2$=$k_3$=$k_4$=$k_5$= $k_6$=1, $k_7$=0,1, and $k_8$=$k_9$=0,01.
Lck-total=Lck+Lck* and Zap70-total=Zap70+Zap70*.

As T cells exert an important control over the immune system, the fine tuning of T cell activity can have great consequences on the responses that immune system triggers against viruses or bacteria, as well as on the development of

autoimmune diseases. We show how a specific molecule type, namely Cbl-b, could tune the threshold required for cell activation. More complex molecular networks that trigger qualitatively different cell responses (full cell activation or anergy) may be considered [5]. These results, together with other wet lab data, may open new perspectives in pharmacological manipulation of immune responses. Drugs may trigger, enhance, diminish or stop the ways in which T cells respond, adjusting the expression level or activity level of the signaling proteins. We consider that simulations of T cell signaling mechanisms could reveal useful informations on immunodeficiencies, autoimmune disorders, vaccine design, as well as the function of healthy immune system.

## 5 Conclusions

P systems were not initially created to model biological systems, although similarities can be observed. Despite many results of universality and several formal language problems which could be explained in an easier and elegant manner, it is useful and desirable to have more connections with the applied computer science and molecular biology. Trying to strengthen these connections, we present a new version of P systems related to the client–server model used for process interaction in computer networks. We use the new version of P systems called Client-Server P Systems to model molecular processes as signaling pathways and T cell activation by using a CSPS software environment called MOlNET. The proposed models for tuning the activation thresholds take into consideration both qualitative and quantitative aspects. We intend to investigate further the proposed CSPS. One goal is to refine CSPS in order to capture more details of the molecular processes.

Many proteins mediate their biological functions through protein interactions. Large networks of such interactions are likely to regulate biological processes rather than single proteins acting by themselves. The benefits of modeling with CSPS in MOlNET are two-fold. First, it has an important role in understanding how an individual protein contribute to global cell behaviour. In this respect, experimental biology is necessary in order to characterize the proteins (if their structure and/or functions are unknown). Second, the number of experiments required to explore all the interactions between many molecules is enormous and would exceed any research budget. Simulations would provide in this case a way to test and search for new partners of interactions for a given protein such as the whole network behaviour would not be affected. The results of the model we described might become even clearer in the context of more global molecular mechanisms of diseases and drug action.

In addition to these, modeling may provide some insights into the complexity of T cell signaling mechanisms. T cell sensitivity and specificity are properties of the signaling network that could be traced out computationally. Models of how different series of signals are coupled to gene expression may explain how one pattern of signaling leads to T cell activation and proliferation, while another leads to T cell unresponsiveness [5]. According to [16], these software cell systems

might have unexpected results and could become the platform on which much biological and medical exploration will be carried out.

## References


1. Bhalla, U., Iyengar, R., *Emergent Properties of Networks of Biological Signaling Pathways*, Science 283, 381-387, 1999.
2. Chan, C., Modeling T cell activation, PhD thesis, Centre for Nonlinear Dynamics and its Applications, University College London, 2002.
3. Ciobanu, G., Paraschiv, D., *P System Software Simulator*, Fundamenta Informaticae 49, 61-66, 2002.
4. Ciobanu, G., Tanasă, B., Dumitriu, D., Huzum, D., Moruz, G., *Molecular Networks as Communicating Membranes*. In Gh.Păun, C.Zandron (Eds.), Proc. Workshop On Membrane Computing, MolCoNet vol.1, 163-175, 2002.
5. Ciobanu, G., Tanasă, B., Dumitriu, D., Huzum, D., Moruz, G., *Simulation and prediction of T cell responses*, Proc. 3rd Intl.Conf. on Systems Biology, 2002.
6. Dassow, J., Păun, Gh., Regulated Rewriting in Formal Language Theory, Springer, Berlin, 1989.
7. Grossman, Z., Singer, A., *Tuning of activation thresholds explains flexibility in the selection and development of T cells in the thymus*, PNAS 93, 14747-14752, 1996.
8. Kaufman, M., Andris, F., Leo, O., *A logical analysis of T cell activation and anergy*, PNAS 96, 3894-3899, 1999.
9. Martín-Vide, C., Păun, A., Păun, Gh., *On the Power of P Systems with Symport Rules*, JUCS 8, 317–331, 2002.
10. Myung, P., Boerthe, N., Koretzky, G., *Adapter proteins in lymphocyte antigen-receptor signaling*, Current Opinion in Immunology 12, 256-266, 2000.
11. Păun, A., Păun, Gh., *The power of communication: P systems with symport/antiport*, New Generation Computing 20, 295–306, 2002.
12. Păun, Gh., *Computing with membranes*, Journal of Computer and System Sciences 61, 108–143, 2000.
13. Rao, N., Dodge, I., Band, H., *The Cbl family of ubiquitin ligases: critical negative regulators of tyrosine kinase signaling in the immune system*, Journal of Leukocyte Biology 71, 753-763, 2002.
14. Rozenberg, G., Salomaa, A., Handbook of Formal Languages, Springer, 1997.
15. Rudd, C., Schneider, H., *Cbl sets the threshold for autoimmunity*, Current Biology 10, 344-347, 2000.
16. Tomita, M., *Whole-cell simulation: a grand challenge of the 21st century*, Trends in Biotechnology 19, 205-210, 2001.

# P Automata or Purely Communicating Accepting P Systems *

Erzsébet Csuhaj-Varjú and György Vaszil

Computer and Automation Research Institute
Hungarian Academy of Sciences
Kende utca 13-17, 1111 Budapest, Hungary
{csuhaj,vaszil}@sztaki.hu

**Abstract.** In this paper we introduce the notion of a P automaton with one-way communication, a concept related both to P systems and the traditional concept of automata. A P automaton with one-way communication is a purely communicating accepting P system. The result of the computation in these systems is the set of multiset sequences consumed by the skin membrane, supposing that the P automaton started functioning in an initial state and entered a so-called final state. As a result, we show that for any recursively enumerable language, a P automaton and a certain type of projection can be constructed such that the given language is obtained as the image of the set of accepted input multiset sequences of the P automaton.

## 1 Introduction

P systems were introduced in [4] as computing models inspired by the functioning of the living cell. Their main components are membrane structures consisting of membranes hierarchically embedded in the outermost skin membrane. Each membrane encloses a region containing a multiset of objects and possibly other membranes. Each region has an associated set of operators operating on the objects contained by the region. These operators can be of different types, they can work on the multisets of objects in the regions but also can provide the possibility of transferring the objects from one region to another one. Since 1998 the theory of P systems or membrane computing has proved to be a successful area of unconventional models of computation: several variants of the basic notion have been introduced and studied proving the power of the framework; the interested reader is referred to [5,6,9] for basic information, and to the book [8] for a summary of the achievements and open problems in the area. The reader also can consult the P systems web page with a lot of downloadable papers and information [11].

By introducing the notion of a P automaton, our paper attempts to build a bridge between the theory of P systems and automata theory, and at the same

* Research supported in part by the Hungarian Scientific Research Fund "OTKA" grants no. T 029615, F 037567, and by the MolCoNet project, IST-2001-32008.

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 219–233, 2003.

time, initiate the study of accepting P systems, systems where the described language is given by the objects entering the structure during the functioning.

According to the original model of a P system, at the beginning of the computation the membranes and the regions of the P system are initialized, that is, they contain multisets of objects and operators. Then, under the given operations, the contents of the regions (the multisets of objects contained in them) change and these changes define the computation. The result of the computation can be, for example, a multiset or set of objects filtered at some previously fixed stage of the computation.

Our model applies another approach. In this case, the membranes of the system are allowed only to communicate with each other, that is, the only allowed activity is to transfer a multiset of objects from one region to another one, supposing that the regions satisfy some prescribed conditions. The skin membrane is allowed to consume multisets of objects from outside, and this is the only way how new objects can enter the membrane system. The computation starts at some initial state of the P system, that is, the initial contents of the different regions are given, and it ends when the P system is in a final state, that is, the regions contain some previously prescribed multisets of objects. The result of the computation is the sequence of multisets of objects which served as input multisets consumed by the skin membrane during an accepting computation, that is, during a computation starting in the initial state and ending in a final state.

It is easy to see that the notion is a related concept to the customary notion of automaton: the input multiset sequence of the P automaton corresponds to the word read from the input tape and the membranes and the regions with the objects represent both storage tapes and states of an automaton. The notion is a natural concept arising from membrane computing: Starting from the initial state, the work of the system is influenced by the effects of the external world. Then it enters a final configuration which can be, for example, a balanced situation or, as in our case, a previously prescribed configuration.

The idea of P automaton was inspired by two problems raised by Gheorghe Păun. The first, from [7] is the following. "What about the possibility of considering a class of P systems, meant to compute, where no rule for objects evolution appears, but only rules governing object communication from a region to another one". The second, from [8], Problem $Q32$: "What about using P systems as accepting devices?"

In this paper we discuss P automata with one-way communication, that is, where each membrane is allowed to ask only its parent membrane (its direct predecessor node in the tree representing the membrane structure of the system) for a multiset of objects (symbols) if a certain condition is satisfied: if the child contains a given multiset of objects. Thus, the communication is of type one-way, top-down. Obviously, a P automaton with two-way communication can also be defined; we shall return to these models in future papers.

The concept of P automata raises several interesting problems. A natural question is the following: What can we say about the accepted input multiset

sequences of a P automaton. We give an answer, and at the same time, demonstrate how these tools can be used for computation: we show that any recursively enumerable language can be represented by a one-way P automaton and a certain type of projection as the image of the set of accepted input multiset sequences. Moreover, the P automaton has only seven membranes. The optimality of this number remains an open problem.

## 2 Definitions

Throughout the paper we assume the reader to be familiar with the basics of language theory; for further details confer [10]. Let $\Sigma$ be an alphabet. Let $\Sigma^*$ be the set of all words over $\Sigma$ (including the empty word $\varepsilon$). We denote the length of a word $w \in \Sigma^*$ by $|w|$, and the number of occurrences of a symbol $a \in \Sigma$ in $w$ by $|w|_a$.

A multiset of objects $M$ is a pair $M = (V, f)$, where $V$ is an arbitrary (not necessarily finite) set of objects and $f$ is a mapping $f : V \to N$ which assigns to each object in $V$ its multiplicity in $M$. The support of multiset $M = (V, f)$ is the set $supp(M) = \{a \in V \mid f(a) \geq 1\}$. The set of all multisets over the set $V$ is denoted by $V^\circ$. If $V$ is a finite set, then $M$ is called a finite multiset. The number of objects in a finite multiset $M = (V, f)$, the cardinality of $M$, is defined by $card(M) = \sum_{a \in V} f(a)$. We say that $a \in M = (V, f)$ if $a \in supp(M)$, and $M_1 = (V_1, f_1) \subseteq M_2 = (V_2, f_2)$ if $supp(M_1) \subseteq supp(M_2)$ and for all $a \in V_1$, $f_1(a) \leq f_2(a)$. The union of two multisets is defined as $(M_1 \cup M_2) = (V_1 \cup V_2, f')$ where for all $a \in V_1 \cup V_2$, $f'(a) = f_1(a) + f_2(a)$, and the difference is defined for $M_2 \subseteq M_1$ as $(M_1 - M_2) = (V_1 - V_2, f'')$ where $f''(a) = f_1(a) - f_2(a)$ for all $a \in V_1 - V_2$. We say that a multiset $M$ is empty, if $supp(M) = \emptyset$.

The reader can easily observe that multiset $M$ over the finite set of objects $V = \{a_1, \ldots, a_n\}$ can be represented as a string $w$ over the alphabet $V$ with $|w|_{a_i} = f(a_i)$, $1 \leq i \leq n$. Clearly, all words obtained from $w$ by permuting the letters can also represent $M$. In the following we will use this type of representation; we will identify the finite multiset of objects $M = (V, f)$ with the word $w$ over the alphabet $V$ representing $M$, thus we will write $w \in V^\circ$, and we will denote the empty multiset as $\varepsilon$.

Let $\Sigma$ and $V$ be two alphabets with $\Sigma \subseteq V$. A mapping $h : V^\circ \to 2^\Sigma$ defined by $h(M) = supp(M) \cap \Sigma$ for a finite multiset $M = (V, f)$ is said to be an $l$-projection of $V^\circ$ to $2^\Sigma$. Mapping $h$ is extended to sequences of finite multisets $M_1 \ldots M_n$, $M_i \in V^\circ$, $1 \leq i \leq n$, in the following way: $h(M_1 \ldots M_n) = \{x_1 \ldots x_n \in \Sigma^* \mid x_i \in h(M_i)$ for $h(M_i) \neq \emptyset$, and $x_i = \varepsilon$ otherwise, $1 \leq i \leq n\}$.

Now we recall some basic notions from membrane computing. For further details the reader is referred to [5,6].

A membrane structure $\mu$ is represented by a Venn diagram and it is identified by a string of matching parentheses with a unique external pair of parentheses. This external pair corresponds to the external membrane called the skin membrane. A membrane structure $\mu$ can also be represented as a tree. The root of the tree corresponds to the skin membrane, and if a node labelled by $k$ is a

direct predecessor of a node labelled by $i$ in the tree, then membrane $i$ is contained in membrane $k$ in the system and there is no other membrane $j$, $j \neq i, k$, such that membrane $i$ is contained in membrane $j$ and membrane $j$ is contained in membrane $k$. In this case we say that membrane $k$ is the parent membrane of membrane $i$ and we use notation $par(i) = k$. Membrane $i$ is called a child membrane of membrane $k$. The skin membrane is always labelled by 1.

As we have already mentioned, each membrane encloses a region, possibly containing other membranes, and also specific objects (used in the multiset sense). Analogously to the membranes, we speak about parent and child regions. The parent region of the skin membrane is the outer region.

In the following we shall define the *contents of a region* as the multiset of all objects that are contained by the region but not contained by any of its child regions. If we consider a multiset of objects which are contained by the region, but not by any of its child regions, then we shall use the term that the region contains this multiset of objects.

Now we define the notion of a one-way P automaton.

**Definition 1.** A *one-way P automaton* with $n$ membranes is a construct $\Gamma = (V, \mu, (w_1, P_1, F_1), \ldots, (w_n, P_n, F_n)), n \geq 1$, where

- $V$ is an alphabet of objects,
- $\mu$ is a membrane structure of $n$ membranes,
- $w_i \in V^\circ, 1 \leq i \leq n$, is the initial contents (state) of region $i$, that is, it is the multiset of all objects contained by region $i$,
- $P_i, 1 \leq i \leq n$, is a finite set of communication rules associated to region $i$, they have the form $(y, in)|_x$ where $x, y \in V^\circ$.
  The rule $(y, in)|_x$ means the following: For $x, y \neq \varepsilon$, if $x$ is contained in region $i$ and $y$ is contained in its parent region, then the objects of $y$ must leave the parent region and enter region $i$. If $x = \varepsilon$, then the region $i$ must be empty, if $y = \varepsilon$, then no object is requested form the parent region.
- $F_i \subseteq V^\circ, 1 \leq i \leq n$, is either a finite set of multisets over $V$ or $F_i = \emptyset$, with $F_j \neq \emptyset$ for at least one $j$, $1 \leq j \leq n$. $F_i$ is called the set of final states of region $i$.

Note that the communication rules we use were introduced as symport rules with promoters in [3]. Note also that for the same condition $x$ more than one requests are allowed, that is, the rule set of region $i$ can have rules $(y, in)|_x$ and $(z, in)|_x$ for $y \neq z$. The productions require two conditions to be satisfied: the multisets represented by $x$ and $y$ (or $z$) must be contained by region $i$ and by the parent region, $par(i)$, respectively.

**Definition 2.** The $n$-tuple of multisets of objects present in the $n$ regions of $\Gamma$ describes the *configuration* of the system; $(w_1, \ldots, w_n)$ is the initial configuration.

Now we define the way of functioning of the one-way P automaton. In each step of the computation, every region asks its parent (the skin membrane asks the outer region) for a multiset of objects by applying one of its rules chosen

non-deterministically. If all requests are satisfied, the system enters a new configuration given by the state (contents) of the $n$ regions. If any of the requests cannot be satisfied, or there is no production to be applied at a region, then the system aborts. Note that the rules are used *sequentially*, one rule per region in each computational step.

**Definition 3.** The transition mapping of a one-way P automaton is a partial mapping $\delta : V^\circ \times (V^\circ)^n \to (V^\circ)^n$. For two configurations $(u_1, \ldots, u_n)$, $(u'_1, \ldots, u'_n)$ and a multiset $u \in V^\circ$,

$$\delta(u, (u_1, \ldots, u_n)) = (u'_1, \ldots, u'_n)$$

holds if for all $i, 1 \leq i \leq n$, there exists a rule $(y_i, in)|_{x_i} \in P_i$ with $x_i \subseteq u_i$ for $x_i \neq \varepsilon$, and $u_i = \varepsilon$ for $x_i = \varepsilon$, and $y_j \subseteq u_{par(j)},\ 2 \leq j \leq n$, $y_1 = u$, and

$$u'_i = u_i \cup y_i - \bigcup_{i=par(j)} y_j. \tag{1}$$

If (1) cannot be satisfied, then $\delta(u, (u_1, \ldots, u_n))$ is undefined. If such a condition is encountered during its funtioning, the system aborts.

The sequence of configurations obtained in the above manner is a computation. The computation ends if the system is in a final state, that is, in a configuration $(\alpha_1, \ldots, \alpha_n)$ having all regions, $1 \leq i \leq n$, with $F_i \neq \emptyset$ in a final state $\alpha_i \in F_i$. If for some $j$, $F_j = \emptyset$, then a configuration can be final regardless of the contents of region $j$.

The reader can observe that the sequence of multisets of objects requested by the skin membrane of the P automaton from the outer region can be considered as an input sequence, and the regions are related to tapes which change their contents in parallel depending on the input. Thus, we define a sequence of multisets of objects accepted by the P automaton as an input sequence which, after being consumed by the skin membrane, causes the system to enter a final state.

**Definition 4.** Let us extend $\delta$ to $\bar{\delta}$, a function mapping the sequences of multisets over $V$ and the configurations $(u_1, \ldots, u_n)$ of $\Gamma$ to new configurations. We define $\bar{\delta}$ as

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. $\bar{\delta}((v_1) \ldots (v_s), (u_1, \ldots, u_n)) = \delta(v_s, \bar{\delta}((v_1) \ldots (v_{s-1}), (u_1, \ldots, u_n)))$, $v_j, u_i \in V^\circ, 1 \leq i \leq n,\ 1 \leq j \leq s$.

Note that we use brackets in the multiset sequence $(v_1) \ldots (v_s)$ in order to distinguish it from the multiset $v_1 \ldots v_s$.

**Definition 5.** Let $\Gamma$ be a one-way P automaton as above. The *language accepted by* $\Gamma$ is the set of multiset sequences

$$L_{acc}(\Gamma) = \{(v_1) \ldots (v_s) \mid \bar{\delta}((v_1) \ldots (v_s), (w_1, \ldots, w_n)) = (u_1, \ldots, u_n) \text{ with } u_j \in F_j \text{ for all } j \text{ with } F_j \neq \emptyset,\ 1 \leq j \leq n, s \geq 1\}.$$

According to this definition, the P automaton is in a final configuration if all of its regions having a set of final states different from the empty set are in one of these final states.

There might be other ways of defining the accepting configuration. To require that only one region is in a final state, is an example of the numerous possibilities.

*Example 1.* Consider the following one-way P automaton.

$$\Gamma = (\{S_1, S_2, S_3, a, b, c\}, [_1\ [_2\ [_3\ ]_3\ ]_2\ ]_1, \\ (S_1, P_1, \{\varepsilon\}), (S_2, P_2, \{S_1S_2\}), (S_3, P_3, \emptyset)),$$

with

$$P_1 = \{(a, in)|_{S_1}, (a, in)|_a, (b, in)|_a, (b, in)|_b, (c, in)|_b, (c, in)|_c, (\varepsilon, in)|_c, (\varepsilon, in)|_\varepsilon\},$$
$$P_2 = \{(S_1, in)|_{S_2}, (a, in)|_{S_1}, (b, in)|_{S_1}, (c, in)|_{S_1}, (\varepsilon, in)|_c\},$$
$$P_3 = \{(\varepsilon, in)|_{S_3}, (abc, in)|_{S_3}\},$$

This P automaton accepts multiset sequences of the form $(a)^n(b)^n(c)^n,\ n \geq 1$. The work of the system starts with the initial configuration $(S_1, S_2, S_3)$, and after the first transition it is found in $(a, S_2S_1, S_3)$. Now by applying the second rules of $P_1, P_2$, and the first rule of $P_3$, the automaton reads a sequence of $a$-s until it reaches the configuration $(a, S_2S_1a^{n-1}, S_3)$ and then $(b, S_2S_1a^n, S_3)$. Now by applying the fourth rule of $P_1$ and the third rule of $P_2$ it reads a sequence of $b$-s, reaching $(b, S_2S_1a^nb^{m-1}, S_3)$, and then $(c, S_2S_1a^nb^m, S_3)$. Now a sequence of $c$-s is read in a similar manner and the automaton reaches $(c, S_2S_1a^nb^mc^{l-1}, S_3)$ and then $(\varepsilon, S_2S_1a^nb^mc^l, S_3)$. Now by using the rule $(abc, in)|_{S_3}$ of $P_3$, the automaton compares the number of letters read before, while also the last rule of $P_1$ and the last rule of $P_2$ is applied. If $n = m = l$ then the automaton reaches a final state $(\varepsilon, S_1S_2, S_3a^nb^mc^l)$.

## 3 Representation of *RE* in Terms of One-Way P Automata

In this section we show that any language that can be recognized by a two-counter machine can also be obtained as an $l$-projection of the language accepted by a one-way P automaton with seven membranes. Since every recursively enumerable language is the accepted language of a two-counter machine, the statement gives a representation of the recursively enumerable language class in terms of P automata.

First we recall the notion of a two-counter machine; for further details the reader is referred to [1,2]. A two-counter machine is a 3-tape Turing machine, $M = (\Sigma \cup \{Z, B\}, Q, R)$ where $\Sigma$ is an alphabet (the alphabet of input symbols), $Q$ is a set of states with two distinguished elements, $q_0, q_f \in Q$, and $R$ is a set of transition rules. The state $q_0$ is called the initial state and $q_f$ is called the final (accepting) state of $M$. The machine has a read only input tape and two semi-infinite storage tapes (the counters). The alphabet of the storage tapes consists of two symbols, $Z$ and $B$ (blank), while the alphabet of the input tape

is $\Sigma \cup \{B\}$. The transition rules of $R$ are of the form $x = \langle b, q, c_1, c_2, q', e_1, e_2, g \rangle$, where $b \in \Sigma \cup \{B\}$ is the symbol scanned on the input tape in state $q \in Q$ and $c_1, c_2 \in \{Z, B\}$ are the symbols scanned on the storage tapes. $M$ enters into state $q' \in Q$, the counters should be modified according to $e_1, e_2 \in \{-1, 0, +1\}$, that is, the counter is incremented by one $(+1)$, the contents of the counter remains unchanged $(0)$, or the counter is decremented by one $(-1)$. The input head moves according to $g \in \{0, +1\}$. If $g = +1$, then the head moves one cell to the right, if $g = 0$, then the head remains in the same position.

Symbol $Z$ appears on the cells initially scanned by the storage tape heads and it never appears on any other cell. An integer $i$ can be stored by moving a storage tape head $i$ cells to the right of $Z$ (the tape contains $ZB^i$). A stored number can be incremented or decremented by moving the tape head right or left. The machine is capable of checking whether a stored value is zero or not, by looking at the symbol scanned by the storage tape heads. If the scanned symbol is $Z$, then the value stored in the corresponding counter is zero.

A word $w \in \Sigma^*$ is accepted by the two counter machine if starting from the initial configuration (having an input word on the input tape, being in the initial state, and reading $Z$s on both of the counter tapes), the two-counter machine enters an accepting configuration, that is, the input head scanned the last non-blank symbol and the machine is in the accepting state.

Two counter machines are computationally complete; they are just as powerful as the Turing machines [1,2].

Now we present our theorem.

**Theorem 1.** *Any recursively enumerable language $L \subseteq \Sigma^*$ can be obtained as an l-projection of the language $L_{acc}(\Gamma)$ of some one-way P automaton $\Gamma$ with seven membranes.*

*Proof.* Let $L \subseteq \Sigma^*$ be a recursively enumerable language accepted by a two-counter machine $M = (\Sigma \cup \{Z, B\}, Q, R)$. We construct a one-way P automaton $\Gamma = (V, \mu, (w_1, P_1, F_1), \ldots, (w_7, P_7, F_7))$ such that the accepted input sequences of $\Gamma$ describe transition sequences of $M$ which starting from an initial configuration, lead to an accepting configuration. The idea of the construction of $\Gamma$ is based on the following considerations.

To simulate the activities required by a transition $x = \langle b, q, c_1, c_2, q', e_1, e_2, g \rangle$ of $M$, dedicated membranes and regions of $\Gamma$ are constructed. The region of the first membrane (the skin membrane) of $\Gamma$ is responsible for simulating the scanning of the actual input symbol, in this case symbol $b$, the second and fourth, respectively the third and fifth regions simulate the checking procedure to establish whether or not the symbol scanned on the counter tape corresponds to $c_1$, respectively $c_2$, and they also simulate the change in the contents of the first and second counters according to $e_1$ and $e_2$, respectively, all determined by the simulated transition. When the automaton starts the simulation of a transition $x$ of $M$, then the second, respectively the third region of $\Gamma$ contains as many $A$ symbols as the value stored in the corresponding counter of $M$ at that moment of the computation. The sixth and the seventh regions are for collecting symbols that are unnecessary for the further computation steps in $\Gamma$. The change

of state $q$ to $q'$ is simulated with the help of special symbols that maintain the synchronized actions of the membranes under the simulation of the execution of transition $x$ in $\Gamma$. These symbols are also for simulating parameter $g$ describing the position of the input tape head. The membrane structure, $\mu$, composed by the seven membranes can be seen on Figure 1.

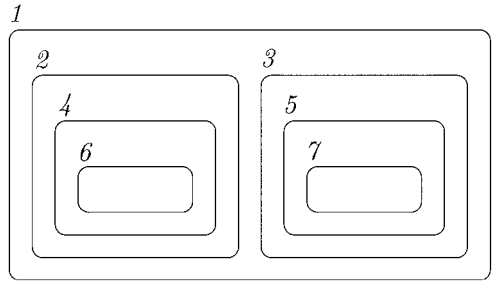



**Fig. 1.** The membrane structure of $\Gamma$.

To help the reader in understanding how the automaton is designed, we explain the simulation of the execution of $x = \langle b, q, c_1, c_2, q', e_1, e_2, g \rangle$, a transition of $M$, with components defined above.

At the beginning of the simulation, the contents of the region of the skin membrane are occurrences of symbols $x$ and $\bar{x}$, which indicate that the simulation of transition $x$ will follow. The second, respectively the third region contains symbols referring to the simulated transition ($F_x$ or $x'$, depending on whether the counter's contents is empty or not) and as many occurrences of $A$ symbols as the number stored in the corresponding counter of $M$. The fourth, respectively the fifth region contains symbols identifying the simulated transition ($x''$) and several $A$s. Finally, the sixth and the seventh region both contain one occurrence of symbol $D$ and possibly several occurrences of symbols which play no further role in the work of the system.

Then $\Gamma$ performs a computation step, where the symbols referring to transition $x$ move from the corresponding region to its child region and an input multiset of $\Gamma$ is consumed from outside by the skin membrane. The successful transition of $\Gamma$ is possible only if the input multiset consists of occurrences of the input symbol $b$ of the simulated transition, $x$.

At the next step, the symbols referring to transition $x$ and the input symbol $b$ of $x$ move one region down and the skin membrane consumes symbols from outside which identify the next transition of $M$ to be applied (if the next transition is $y$, then the the symbols are $y''$-s) and as many occurrences of $A$s as it is necessary to perform the modification of the contents of the counters of $M$ according to parameters $e_1$ and $e_2$ of the simulated transition $x$. This is the step where the check whether or not the contents of the counters correspond to $c_1$ and $c_2$ is performed. If $c_1, c_2$ prescribes a nonempty counter, then a symbol $A$ has to move from the second, respectively from the third region to the fourth, respectively

to the fifth region. If $c_1, c_2$ prescribes an empty counter, then the move of the input symbol $b$ and that of symbol $\overline{x}$ from the region of the skin membrane to the second, respectively to the third region is possible only if the region is empty. These communication steps can only be performed if the conditions necessary to execute transition $x$ hold.

After the successful transition in $\Gamma$, the skin membrane consumes further symbols (these are $y'$s) from outside which identify the next transition of $M$ (which is $y$) to be performed and the second and the third regions receive as many occurrences of $A$s as it is necessary to perform the modification of the contents of the counters according to $e_1$ and $e_2$. Furthermore, symbols $b$ and $\overline{x}$ from the second, respectively from the third region move to the fourth, respectively to the fifth region. The symbols referring to transition $x$ move from the fourth, respectively from the fifth region move to the sixth, respectively to the seventh region. This step can be performed if the previous steps had been successfully executed.

Finally, at the last step, symbols $y$ and $\overline{y}$ are consumed from outside by the skin membrane to indicate that a new transition, $y$, will be simulated, and symbols $y'$ and $y''$ move to the second, respectively, to the third region and to the fourth, respectively to the fifth region; symbols $b$ and $\overline{x}$ move to the sixth and the seventh region.

The obtained configuration of $\Gamma$ is of the same form as it was at the beginning of the simulation of transition $x$. The move of the reading head of the input tape $(g)$ is taken into account at the initialization of the simulation of the following transition. Obviously, in addition to the rules associated to the transitions of $M$, the regions of $\Gamma$ contain other rules to guarantee the correct simulation.

Now we define the P automaton $\Gamma$ in details. We first note that instead of simulating the functioning of $M$, we construct $\Gamma$ simulating the functioning of the two-counter machine $M'$ which accepts $\$_1 L \$_2$, where $\$_1, \$_2$ are two new symbols not in $\Sigma$.

$M'$ has the states of $M$ and it has some further new states. It is with initial state $q_s$, with final state $q_{fin}$, both different from the initial state and the final state of $M$. Moreover, in addition to $q_f$ $M'$ has a new state $q'_f$. The transition set of $M'$ consists of the transitions of $M$ and some further transitions given as follows: $\langle \$_1, q_s, Z, Z, q_0, 0, 0, 1\rangle$, $\langle \$_2, q'_f, c_1, c_2, q_{fin}, 0, 0, 0\rangle$, where $c_1, c_2 \in \{B, Z\}$, and for each transition $\langle b, q, c_1, c_2, q_f, e_1, e_2, g\rangle$ in $M$ there is a transition in $M'$ of the form $\langle b, q, c_1, c_2, q'_f, e_1, e_2, 1\rangle$. Thus, $M'$ first reads symbol $\$_1$ and then continues reading word $w \in \Sigma^*$ in the same way as $M$, then it reads the symbol $\$_2$ and enters the accepting state $q_{fin}$ if $M$ enters the accepting state $q_f$.

For technical reasons, when simulating the transitions of $M'$ with $\Gamma$, we have to indicate whether or not the reading input head of $M'$ was at the same position at the previous transition. For this purpose we define a new alphabet $\Sigma^e = \{b_e \mid b \in \Sigma\}$.

Let us denote $\Sigma_T = \{x \mid x = \langle b, q, c_1, c_2, q', e_1, e_2, g\rangle \text{ is a transition of } M\}$. Let $x = \langle b, q, c_{1,x}, c_{2,x}, q', e_{1,x}, e_{2,x}, g_x\rangle$ be a transition in $M$ such that $g_x = 0$, that is, the reading head at the input tape does not move after performing the

transition, and suppose that $y = \langle b, q', c_{1,y}, c_{2,y}, q'', e_{1,y}, e_{2,y}, g_y \rangle$ is a transition of $M$. Then, for each transition $y$ with this property we add a new transition $y_e = \langle b_e, q', c_{1,y}, c_{2,y}, q'', e_{1,y}, e_{2,y}, g_y \rangle$ to the set of transitions of $M'$. Let us denote by $\Sigma_T^e$ the set of these new transitions.

We say that a transition $y = \langle b, q_y, c_{1,y}, c_{2,y}, q'_y, e_{1,y}, e_{2,y}, g_y \rangle$ of $M'$ is applicable after transition $x = \langle a, q_x, c_{1,x}, c_{2,x}, q'_x, e_{1,x}, e_{2,x}, g_x \rangle$ in $M'$ if $q'_x = q_y$ holds, and if $g_x = 0$, then $b = a_e$, or if $a \in \Sigma^e$, then $b = a$. Note, that applicability in this sense does not necessarily mean that $y$ can always be executed after $x$ during a computation. It only refers to the fact that $x$ leaves $M'$ in a state, $q'$, which is a necessary, but not sufficient condition for the possibility of executing $y$.

Now we define the alphabet of $\Gamma$ as $V = \Sigma \cup \Sigma_T \cup \Sigma'_T \cup \Sigma''_T \cup \overline{\Sigma}_T \cup \Sigma^e \cup \Sigma_T^e \cup \Sigma_T^{e\prime} \cup \Sigma_T^{e\prime\prime} \cup \overline{\Sigma_T^e} \cup \{F_x \mid x \in \{\Sigma_T \cup \Sigma_T^e\}\} \cup \{A, D, \$_1, \$_2\}$, where $\overline{\Sigma}_T$, $\Sigma'_T$, ${\Sigma_T}''$ $\overline{\Sigma_T^e}$, $\Sigma_T^{e\prime}$, $\Sigma_T^{e\prime\prime}$ are the over-lined, primed and double-primed versions of the corresponding alphabets.

The membrane structure of $\Gamma$, $\mu = [_1\,[_2\,[_4\,[_6\,]_6\,]_4\,]_2\,[_3\,[_5\,[_7\,]_7\,]_5\,]_3\,]_1$, can be seen in Figure 1.

The initial states of the regions are as follows: $w_1 = \overline{x_s x_s}$, $w_2 = w_3 = {x_s}' F_{x_s}$, $w_4 = w_5 = {x_s}''$, where $x_s = \langle \$_1, q_s, Z, Z, q_0, 0, 0, 1 \rangle$, the initial transition of $M'$, and $w_6 = w_7 = D$.

We define the final states of the regions as follows. $F_1$ consists of the following multisets: $\$_2\$_2\overline{y}\overline{y}$ for $y = \langle \$_2, q'_f, c_1, c_2, q_{fin}, 0, 0, 0 \rangle$, the final transition of $M'$ with $c_1, c_2 \in \{B, Z\}$, and $F_i = \emptyset$ for $2 \leq i \leq 7$.

Thus, $\Gamma$ starts its work by consuming an input multiset with symbols $\$_1$ from outside and stops after consuming an input multiset with symbols $\$_2$. Now we define the rule sets of the different membranes.

For each transition $x = \langle b, q, c_{1,x}c_{2,x}, q', e_{1,x}, e_{2,x}, g_x \rangle \in \Sigma_T \cup \Sigma_T^e$, all productions of the forms

$$(bb, in)|_\alpha \in P_1, \text{ where } \alpha = \begin{cases} xx\overline{x}\overline{x} & \text{if } Z \notin \{c_{1,x}, c_{2,x}\}, \\ x\overline{x}\overline{x} & \text{if } Z \in \{c_{1,x}, c_{2,x}\}, c_{1,x} \neq c_{2,x}, \\ \overline{x}\overline{x} & \text{if } c_{1,x} = c_{2,x} = Z. \end{cases}$$

These rules are for checking whether the input symbols from $\Sigma$ consumed from outside by the skin membrane of $\Gamma$ correspond to the input symbol of $M'$ prescribed by the corresponding transitions.

Now, if $y = \langle a, q', c_{1,y}, c_{2,y}, q'', e_{1,y}, e_{2,y}, g_y \rangle \in \Sigma_T \cup \Sigma_T^e$ is an applicable (in the sense defined above) transition after $x$, then

$$(y''y''wz, in)|_{bb\overline{x}\overline{x}} \in P_1$$

where if $c_{1,x} \neq Z$, then $w = AA$ for $e_{1,x} = 1$, $w = A$ for $e_{1,x} = 0$, $w = \varepsilon$ for $e_{1,x} = -1$; if $c_{1,x} = Z$, then $w = A$ for $e_{1,x} = 1$ and $w = \varepsilon$ for $e_{1,x} = 0$; and similarly if $c_{2,x} \neq Z$, then $z = AA$ for $e_{2,x} = 1$, $z = A$ for $e_{2,x} = 0$, $z = \varepsilon$ for $e_{2,x} = -1$; if $c_{2,x} = Z$, then $z = A$ for $e_{2,x} = 1$ and $z = \varepsilon$ for $e_{2,x} = 0$.

Furthermore, if $y$ is as above, then

$$\left.\begin{array}{ll} (y'y', in)|_{y''y''} \text{ and } (yy\overline{y}\overline{y}, in)|_{y'y'} & \text{if } Z \notin \{c_{1,y}, c_{2,y}\}, \\ (y'y'F_y, in)|_{y''y''} \text{ and } (y\overline{y}\overline{y}, in)|_{y'y'} & \text{if } Z \in \{c_{1,y}, c_{2,y}\}, c_{1,y} \neq c_{2,y}, \\ (y'y'F_yF_y, in)_{y''y''} \text{ and } (\overline{y}\overline{y}, in)|_{y'y'} & \text{if } c_{1,y} = c_{2,y} = Z, \end{array}\right\} \in P_1.$$

These rules are for initializing the simulation of each transition, $y$, which is applicable after the transition $x$ of $M'$.

Rule sets $P_2$ and $P_3$ are defined in the same way. For each transition $x = \langle b, q, c_{1,x}, c_{2,x}, q', e_{1,x}, e_{2,x}, g_x \rangle \in \Sigma_T \cup \Sigma_T^e$, let

$$\left.\begin{array}{ll} (x, in)|_{x'} & \text{if } c_{i-1,x} \neq Z, \\ (\varepsilon, in)|_{x'F_x} & \text{if } c_{i-1,x} = Z, \end{array}\right\} \in P_i,\ 2 \leq i \leq 3.$$

These rules assist to synchronizing the activity of the regions while simulating a transition in $M'$. Let also

$$\left.\begin{array}{ll} (b\overline{x}, in)|_{x} & \text{if } c_{i-1,x} \neq Z, \\ (b\overline{x}, in)|_{\varepsilon} & \text{if } c_{i-1,x} = Z, \end{array}\right\} \in P_i,\ 2 \leq i \leq 3.$$

These rules assist in checking whether or not the actual contents of the counters of $M'$ are represented in the second, respectively in the third region in an appropriate manner.

For each $y$ applicable after a transition $x$ as above, let

$$(y''w, in)|_b \in P_i,\ 2 \leq i \leq 3,$$

where $w = AA$ if $e_{i-1,x} = 1$, $w = A$ if $e_{i-1,x} = 0$, and $w = \varepsilon$ if $e_{i-1,x} = -1$ for $c_{i-1,x} \neq Z$; $w = A$ if $e_{i-1,x} = 1$, and $w = \varepsilon$ if $e_{i-1,x} = 0$ for $c_{i-1,x} = Z$. Let also

$$\left.\begin{array}{ll} (x', in)|_{x''} & \text{if } c_{i-1,x} \neq Z, \\ (x'F_x, in)|_{x''} & \text{if } c_{i-1,x} = Z, \end{array}\right\} \in P_i,\ 2 \leq i \leq 3.$$

for each transition $x = \langle b, q, c_{1,x}, c_{2,x}, q', e_{1,x}, e_{2,x}, g_x \rangle \in \Sigma_T \cup \Sigma_T^e$. These rules assist in initializing the simulation of the next transition to be performed after a transition in $M'$.

Rule sets $P_4$ and $P_5$ are defined in the same way. For each transition $x = \langle b, q, c_{1,x}, c_{2,x}, q', e_{1,x}, e_{2,x}, g_x \rangle \in \Sigma_T \cup \Sigma_T^e$, let

$$\left.\begin{array}{ll} (x', in)|_{x''} & \text{if } c_{i-3,x} \neq Z, \\ (x'F_x, in)|_{x''} & \text{if } c_{i-3,x} = Z, \end{array}\right\} \in P_i,\ 4 \leq i \leq 5.$$

These rules contribute to maintaining the synchronized activity of the regions when simulating a certain transition, $x$, in $M'$.

Let also

$$\left.\begin{array}{ll} (Ax, in)|_{x'} & \text{if } c_{i-3,x} \neq Z, \\ (\varepsilon, in)|_{x'F_x} & \text{if } c_{i-3,x} = Z, \end{array}\right\} \in P_i,\ 4 \leq i \leq 5.$$

These rules take part in checking whether or not the actual contents of the counters of $M'$ are represented in the second, respectively in the third region in an appropriate manner. Furthermore, let

$$\left.\begin{array}{ll} (b\overline{x}, in)|_{x} & \text{if } c_{i-3,x} \neq Z, \\ (b\overline{x}, in)|_{F_x} & \text{if } c_{i-3,x} = Z, \end{array}\right\} \in P_i,\ 4 \leq i \leq 5,$$

and also

$$(x'', in)|_b \in P_i,\ 4 \leq i \leq 5.$$

These rules help in maintaining the synchronized activity of the regions and also take part in initializing the simulation of the next transition to be performed.

Rule sets $P_6$ and $P_7$ are defined in the same way. For each transition $x = \langle b, q, c_{1,x}, c_{2,x}, q', e_{1,x}, e_{2,x}, g_x \rangle \in \Sigma_T \cup \Sigma_T^e$, let

$$(x'', in)|_D,\ (x', in)|_D,\ (b\overline{x}, in)|_D \in P_i,\ 6 \leq i \leq 7,$$

and also

$$\left.\begin{array}{ll} (x, in)|_D & \text{if } c_{i-5,x} \neq Z, \\ (F_x, in)|_D & \text{if } c_{i-5,x} = Z, \end{array}\right\} \in P_i,\ 6 \leq i \leq 7.$$

These rules assist in maintaining the synchronized activity of the regions under simulating a certain transition of $M'$ by collecting symbols unnecessary for further computation steps.

Now we prove that transition sequences of $\Gamma$ correspond to transition sequences of $M'$. In order to help the reader understand our reasoning, in the following we denote the configurations of $\Gamma$ by giving the bracket notation for the membrane structure with the regions containing the appropriate multisets. We also use a similar bracket notation when giving the 7-tuples of rules to be applied at certain steps of the computation, replacing the contents of the regions by the appropriate rules.

Suppose that at some stage of the computation, $\Gamma$ is in configuration

$$[_1\ t\ [_2\ t'_1\alpha\ [_4\ x''\gamma\ [_6\ Du\ ]_6]_4]_2\ [_3\ t'_2\beta\ [_5\ x''\delta\ [_7\ Dv\ ]_7]_5]_3]_1 \tag{2}$$

where $t = xx\overline{xx}$ if $Z \notin \{c_{1,x}, c_{2,x}\}$, $t = x\overline{xx}$ if $Z \in \{c_{1,x}, c_{2,x}\}$, $c_{1,x} \neq c_{2,x}$, and $t = \overline{xx}$ if $c_{1,x} = c_{2,x} = Z$, for some $x = \langle b, q, c_{1,x}, c_{2,x}, q', e_{1,x}, e_{2,x}, g_x \rangle \in \Sigma_T \cup \Sigma_T^e$. Furthermore, $t'_i = x'$ if $c_{i,x} \neq Z$, $t'_i = x'F_x$ if $c_{i,x} = Z$, $1 \leq i \leq 2$, and $\alpha, \beta, \gamma, \delta \in A^*$, $u, v \in\ (V - \{A, D\})^*$, where $\alpha$ and $\beta$, contain as many occurrences of $A$ as the value stored in the first, respectively in the second counter of $M'$ before the execution of transition, $x$.

Thus, the contents of the first region refers to the transition $x$ of $M'$ which will be simulated, and this plan is also indicated by the second, the third, the fourth and the fifth region. As we have said before, the sixth and the seventh regions collect the symbols which are unnecessary for the further steps of the computation.

In this configuration the only 7-tuple of rules that can be applied is the following.

$$[_1\ (bb, in)_t\ [_2\ (s_1, in)_{t'_1}\ [_4\ (t'_1, in)_{x''}\ [_6\ (x'', in)_D\ ]_6]_4]_2 \\ [_3\ (s_2, in)_{t'_2}\ [_5\ (t'_2, in)_{x''}\ [_7 (x'', in)_D\ ]_7]_5]_3]_1$$

where $t$, $t'_1, t'_2$, and $b$ are as above, and $s_i = x$ if $c_{i,x} \neq Z$, $s_i = \varepsilon$ if $c_{i,x} = Z$, $1 \leq i \leq 2$.

By the application of these rules, the skin membrane checks whether the input multiset identifies the input symbol $b$ of transition $x$, and makes preparations for checking whether or not the contents of the second and the third region correspond to the required contents of the corresponding counters of $M'$ to

successfully perform transition $x$. If rule $(bb, in)_t$ cannot be applied, then the computation aborts.

After successfully applying the above 7-tuple of rules, $\Gamma$ enters the configuration

$$[_1\ bb\overline{xx}\ [_2\ \alpha''\ [_4\ \gamma'\ [_6\ x''Du\ ]_6]_4]_2\ [_3\ \beta''\ [_5\ \delta'\ [_7\ x''Dv\ ]_7]_5]_3]_1$$

where $\alpha'' = x\alpha$ for $c_{1,x} \neq Z$, and $\alpha'' = \alpha$ for $c_{1,x} = Z$; $\beta'' = x\beta$ for $c_{2,x} \neq Z$, $\beta'' = \beta$ for $c_{2,x} = Z$; $\gamma' = x'\gamma$ for $c_{1,x} \neq Z$, and $\gamma' = x'F_x\gamma$ for $c_{1,x} = Z$; $\delta' = x'\delta$ for $c_{2,x} \neq Z$, and $\delta' = x'F_x\delta$ for $c_{2,x} = Z$.

Thus, the second, respectively the third region contains as many occurrences of $A$ as the number stored in the first, respectively in the second counter of $M'$ at this step of the computation. Moreover, if a counter is not empty, and only in this case, the corresponding region contains an occurrence of $x$.

Now the only 7 tuple of rules which can be applied is as follows.

$$[_1\ (y''y''wz, in)|_{bb\overline{xx}}\ [_2\ (b\overline{x}, in)|_{r_1''}\ [_4\ (s_3, in)|_{r_3''}\ [_6\ (x', in)_D\ ]_6]_4]_2 \\ [_3\ (b\overline{x}, in)|_{r_2''}\ [_5\ (s_4, in)|_{r_4''}\ [_7\ (x', in)|_D\ ]_7]_5]_3]_1$$

where $y$ is the next transition to be simulated, $r_i'' = x$ for $c_{i,x} \neq Z$ and $r_i'' = \varepsilon$ for $c_{i,x} = Z$, $1 \leq i \leq 2$; $r_j'' = x'$ for $c_{j-2,x} \neq Z$, and $r_j'' = x'F_x$ for $c_{j-2,x} = Z$; $s_j = Ax$ for $c_{j-2,x} \neq Z$, and $s_j = \varepsilon$ for $c_{j-2,x} = Z$, $3 \leq j \leq 4$. Moreover, we know that for $y = \langle a, q', c_{1,y}, c_{2,y}, q'', e_{1,y}, e_{2,y}, g_y \rangle$, $a = b_e$ holds if $g_x = 0$, or if $a \in \Sigma^e$, then $a = b$.

These rules simulate the checking of the applicability of transition $x$, and initialize the simulation of the next transition, $y$. The initialization is done by applying rule $(y''y''wz, in)|_{bb\overline{xx}}$ in the first region, where $w, z \in A^*$ with an appropriate number of $A$-s, depending on transition $x$ as described in the definition of the rule set $P_1$.

The checking of the contents of the counters is simulated by applying rules of the second, third, fourth and fifth regions as follows. If the first, respectively the second counter of $M'$ must be empty according to the transition, then the second region, respectively the third region must be empty. This is the only case when rule $(b\overline{x}, in)|_\varepsilon$ can be successfully applied. If the region is not empty, then it contains several $A$-s without any other symbol, so no rule in $P_2$ (in $P_3$) can be applied, thus, the computation aborts.

If the first, respectively the second counter must not be empty, then the second, respectively the third region must contain the transition symbol $x$ and at least one occurrence of $A$, and this is the only case when the rule $(Ax, in)|_{x'}$ can be successfully applied in the fourth, respectively in the fifth region. Otherwise, the computation of $\Gamma$ ends with abnormal termination.

After applying this 7-tuple of rules, the obtained configuration of $\Gamma$ is as follows:

$$[_1\ y''y''wz\ [_2\ b\overline{x}\alpha'''\ [_4\ \gamma''\ [_6\ x'x''Du\ ]_6]_4]_2\ [_3\ b\overline{x}\beta'''\ [_5\ \delta''\ [_7\ x'x''Dv\ ]_7]_5]_3]_1$$

where $\alpha''' \in A^*, |\alpha'''| = |\alpha| - 1$ for $c_{1,x} \neq Z$, and $\alpha''' = \varepsilon$ for $c_{1,x} = Z$; $\beta''' \in A^*, |\beta'''| = |\beta| - 1$ for $c_{2,x} \neq Z$, and $\beta''' = \varepsilon$ for $c_{2,x} = Z$; $\gamma'' = xA\gamma$ for $c_{1,x} \neq Z$, and $\gamma'' = F_x\gamma$ for $c_{1,x} = Z$; $\delta'' = xA\delta$ for $c_{2,x} \neq Z$, and $\delta'' = F_x\delta$ for $c_{2,x} = Z$.

Thus, the first region contains two occurrences of the symbol $y''$ which identify the next transition of $M'$ to be simulated. The region of the skin membrane contains occurrences of $A$ calculated as follows. If the respective counter had to be nonempty before performing the transition, then this region contains one occurrence of $A$ more than it is necessary to add to obtain the new counter contents; if the counter had to be empty, then the number of occurrences of $A$ is the same as the difference of the desired new and the actual counter contents prescribed by the transition. This is calculated for both counters of $M'$.

Then, the next 7 tuple of rules which can be applied is the following.

$$[_1\ (t'_1t'_2, in)|_{y''y''}\ [_2\ (y''w, in)|_b\ [_4\ (b\overline{x}, in)|_{r'''_3}\ [_6\ (r'''_3, in)|_D\ ]_6]_4]_2 \\ [_3\ (y''z, in)|_b\ [_5\ (b\overline{x}, in)|_{r'''_4}\ [_7\ (r'''_4, in)|_D\ ]_7]_5]_3]_1$$

where $t'_i = y'$ for $c_{i,y} \neq Z$, and $t'_i = y'F_y$ for $c_{i,y} = Z$, $1 \leq i \leq 2$; $r'''_3 = x$ for $c_{1,x} \neq Z$, and $r'''_3 = F_x$ for $c_{1,x} = Z$; $r'''_4 = x$ for $c_{2,x} \neq Z$, and $r'''_4 = F_x$ for $c_{2,x} = Z$.

Then the contents of the second region and that of the third region are modified to correspond to the contents of the respective counters of $M'$ after the successful application of transition $x$ and the initialization of the simulation of the next transition, $y$, continues.

The new configuration of $\Gamma$ is as follows.

$$[_1\ t'_1t'_2\ [_2\ y''\alpha^{iv}\ [_4\ b\overline{x}\gamma^{iv}\ [_6\ r_6x'x''Du\ ]_6]_4]_2\ [_3\ y''\beta^{iv}\ [_5\ b\overline{x}\delta^{iv}\ [_7\ r_7x'x''Dv\ ]_7]_5]_3]_1$$

where $\alpha^{iv} \in A^*$ with as many occurrences of $A$ as the value stored in the first counter of $M'$ after performing transition $x$; $\beta^{iv} \in A^*$, with as many occurrences of $A$ as the value stored in the second counter of $M'$ after performing transition $x$; $\gamma^{iv}$ and $\delta^{iv}$ consist of several occurrences of $A$s; $r_6 = x$ for $c_{1,x} \neq Z$, and $r_6 = F_x$ for $c_{1,x} = Z$; $r_7 = x$ for $c_{2,x} \neq Z$, and $r_7 = F_x$ for $c_{2,x} = Z$.

The simulation of transition $x$ and the initialization of the simulation of transition $y$ is finished by applying the 7-tuple of rules

$$[_1\ (t, in)|_{t'_1t'_2}\ [_2\ (t'_1, in)|_{y''}\ [_4\ (y'', in)|_b\ [_6\ (b\overline{x}, in)|_D\ ]_6]_4]_2 \\ [_3\ (t_2, in)|_{y''}\ [_5\ (y'', in)|_b\ [_7\ (b\overline{x}, in)|_D\ ]_7]_5]_3]_1$$

where $t'_1$ and $t'_2$ are defined the same way for transition $y$ as in (2) for transition $x$, and also similarly to (2), $t = yy\overline{yy}$ for $Z \notin \{c_{1,y}, c_{2,y}\}$; $t = y\overline{yy}$ for $Z \in \{c_{1,y}, c_{2,y}\}$, $c_{1,y} \neq c_{2,y}$; and $t = \overline{yy}$ for $c_{1,y} = c_{2,y} = Z$. Moreover, this is the only 7-tuple of rules that can be applied at this step of the computation.

As a result, $\Gamma$ enters configuration

$$[_1\ t\ [_2\ t'_1\alpha^{iv}\ [_4\ y''\gamma^{iv}\ [_6\ b\overline{x}r_6x'x''Du\ ]_6]_4]_2\ [_3\ t'_2\beta^{iv}\ [_5\ y''\delta^{iv}\ [_7\ b\overline{x}r_7x'x''Dv\ ]_7]_5]_3]_1$$

which is of the same form as (2), the configuration we have started from, the simulation of the functioning of $M'$ can continue in the same way.

If at some point the final transition, $x_s$, is simulated, then $\Gamma$ enters a final state, thus, for any accepting computation in $M'$ there is an accepting computation in $\Gamma$ and reversely.

If we consider the $l$-projection $h : V^\circ \to 2^\Sigma$, then we can see that for any accepted input multiset sequence $(a_1) \dots (a_n) \in L(\Gamma)$, $h((a_1) \dots (a_n)) = w \in \Sigma^*$ where $w \in L$. This proves our theorem. □

## 4 Final Remarks

The aim of our paper was to introduce the notion of a P automaton, a P system defined as an accepting device using communication rules only, according to two of the problems raised in [7] and in [8]. Obviously, similarly to the large number of different types of automata, a wide variety of P automata can be defined and studied, with different types of communication or different types of conditions for ending the computation. Apart from the possible variants, an other interesting problem area would be to define complexity measures for these devices. We shall return to these topics in future papers.

## References

1. Fischer, P.C.: Turing machines with restricted memory access. Information and Control **9** (1966) 364–379.
2. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading, Mass., 1979.
3. 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.
4. Păun, Gh.: Computing with membranes. TUCS Report 208, Turku Centre for Computer Science, 1998.
5. Păun, Gh.: Computing with membranes. An introduction. Bulletin of the EATCS **67** (1999) 139–152.
6. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences **61**(1) (2000) 108–143.
7. Păun, Gh.: Computing with membranes (P Systems): Twenty six research topics. CDMTCS Technical Report 119, Univ. of Auckland, 2000.
8. Păun, Gh.: Membrane Computing. An Introduction. Springer-Verlag, Berlin, 2002.
9. Păun, Gh., Rozenberg, G.: A guide to membrane computing. Theoretical Computer Science **287** (2002) 73–100.
10. Rozenberg, G., Salomaa, A., eds.: Handbook of Formal Languages. Volumes I-III. Springer-Verlag, Berlin, 1997.
11. The P systems web page at `http://psystems.disco.unimib.it`

# Self–activating P Systems

Eugen Czeizler

Turku Centre for Computer Science and
Department of Mathematics, University of Turku
Turku FIN 20014, Finland
eugenc@cs.utu.fi

**Abstract.** Until now, the solving of NP complete problems in polynomial time in the framework of P systems was accomplished by the use of three different techniques: the duplication of membranes, the creation of membranes, and the replication of strings. In this paper we introduce a new type of P systems which comes with a new technique of approaching this class of problems. In the initial configuration of these P systems we have an arbitrarily large number of unactivated base-membranes, which, in a polynomial time, are activated in an exponential number. Using these type of systems we solve the SAT problem in a linear time, with respect to the number of variables and clauses.

## 1 Introduction

The P systems are a class of distributed parallel computing models inspired from the way nature organizes the cellular level in living organisms. These systems can be seen as a general computing architecture where various types of objects can be processed by various operations.

A P system, as this was first introduced in [6] by Gh. Paun, consists of a membrane structure (usually, represented graphically by an Euler-Venn diagram and mathematically by a string of labelled parentheses, indicating the membranes and their relative position), consisting of several cell-like membranes which are hierarchically embedded in a main membrane, called the skin membrane. The membranes delimit regions, where we place objects; the objects can be seen as symbols from a given alphabet, or, in some types of P-systems (such as splicing P systems and P systems based on rewriting) they can be strings over an alphabet.

The objects evolve according to given evolution rules, which are associated with the regions delimited by the membrane structure. Each rule can be applied only to the objects placed in those regions associated with the rules, being able to modify the objects, send them either outside the current membrane or to an inner membrane (operation which is called *communication*), and also being able to dissolve the membrane. When such an action takes place, all the objects of the dissolved membrane remain free in the membrane placed immediately outside, but the evolution rules of the dissolved membrane are removed. The skin membrane is never dissolved.

The evolution rules are used in the maximally parallel manner (at each step, all objects which can evolve according to a specific rule should evolve), choosing

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 234–246, 2003.

non-deterministically the rules and the objects to which they are applied. In this way, we obtain a *transition* from a configuration of the system to the next configuration; a sequence of transitions forms a *computation*. A computation is considered *completed* when it halts, i.e., no further rules can be applied to any of the objects present in the last configuration. There are two possible ways of assigning a result to a computation: by considering the multiplicity of all objects present in a designated membrane in a halting configuration, or by concatenating the symbols which leave the system, in the order they are sent out of the skin membrane. Thus, in the first case we compute vectors of natural numbers, while in the second case we generate a language.

It is well-known that a lot of practical problems reduce to NP-complete problems, which are intractable for the usual computers. The parallelism is a possible way to deal with this situation. On the other hand, one of the main features of the membrane systems is their inherent parallelism. In the case of three different classes of P systems (with an enhanced parallelism), NP-complete problems can be solved in polynomial (and even linear) time. These three types (together with some of the papers describing the specific methods of approaching NP-complete problems) are the following: duplication of membranes (in [7], [2], [5]), replication of string-objects (in [1], [3]), and membrane creation (in [4]).

In this paper we introduce a new variant of P systems, together with a new method of approaching NP-complete problems in the framework of membrane computing. The basic idea is that instead of producing in linear (or polynomial) time an exponential work space, we start from the beginning with an exponential, but unused, potential work space in the form of an arbitrarily large number of inactivated basic membranes (we call them base membranes), and in a linear time we activate those membranes and use them in our computation. Thus, the computing will start in the initial configuration within a small amount of active base membranes, and, in a linear time, the objects present in those membranes will activate according to given rules an exponential number of inactivated base membranes. Using this type of systems we solve the SAT problem in a linear time, with respect to the number of variables and clauses. It must be stressed out that the time needed for creating (or putting together) the arbitrarily large number of base membranes is not considered.

## 2 Self-activating P Systems

In the following we define the type of P systems mentioned above, and start by describing its particularities.

We define a *base cell* as being a basic P system, with its own membrane structure, sets of evolution rules, and objects associated with the regions delimited by it. We call the skin membrane of such a base cell the *base membrane*. In general, it is recommended that the membrane structure of such a base cell to be simple, that is with just a few inner membranes.

We consider a base cell to be *inactive*, if, since the beginning of the evolution of the system, no rule could have been applied inside the cell, due to the absence of needed objects. Otherwise, we say that the base cell is *active*.

In a *self-activating* P system we have a skin membrane, inside of which one can find an arbitrarily large number of identically base cells, having the same membrane structure and sets of rules, but not the same objects placed in their regions. From these cells, some are active, and some are inactive; the latter ones can be activated during the following transitions. Such a P system is described in Figure 1.

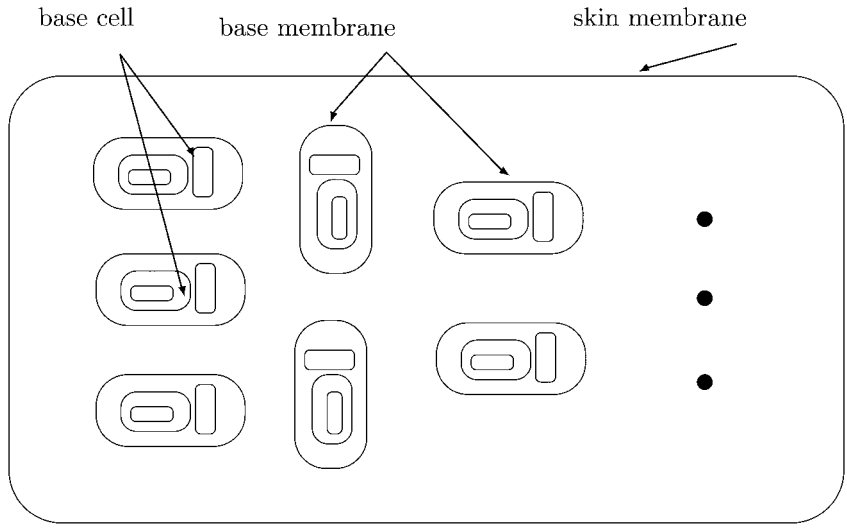



**Fig. 1.** A self-activating P system

This construction is natural, having in consideration the structure of some living organs such as the kidneys, which are composed of thousands of identically cells, the nephrons, which are functioning together as a whole, but not necessarily always all of them simultaneously.

For the rest of the paper we consider that the structure of the base cell is as basic as it could be, that is, it consists of a single membrane, the base membrane. Therefore, in the base cell we have now only one region, and a single set of rules, while the membrane structure of the entire P system consists of a skin membrane, and an arbitrarily large number of base membranes (active or inactive). In order to simplify the notation, we omit the word '*base*', and, when referring to the system we use just the word membrane.

The membranes can have electrical charges (positive or negative), in which case they are marked with $+$ or $-$, or, they can be electrically neutral, in which case they are marked with 0. Also, in order to simplify the notation, when a membrane is neutrally charged and by applying a rule its charge is not changed, we omit the 0 mark.

In the initial configuration, all the inactive membranes "look" the same (we call this form the *initial state*). The initial state is characterized by the electrical charge of the membrane (positive, negative, or neutral), and by the objects present within. Therefore, in the initial configuration, as well as in all next configurations, inside the skin membrane there is an arbitrarily large number of membranes in the initial state (that is, inactive), and a number of active membranes. The latter ones are also called *essential* membranes, because these are the membranes in which the computation really takes place. For example, a configuration of a self-activating P system at a given time could be as shown in Figure 2.

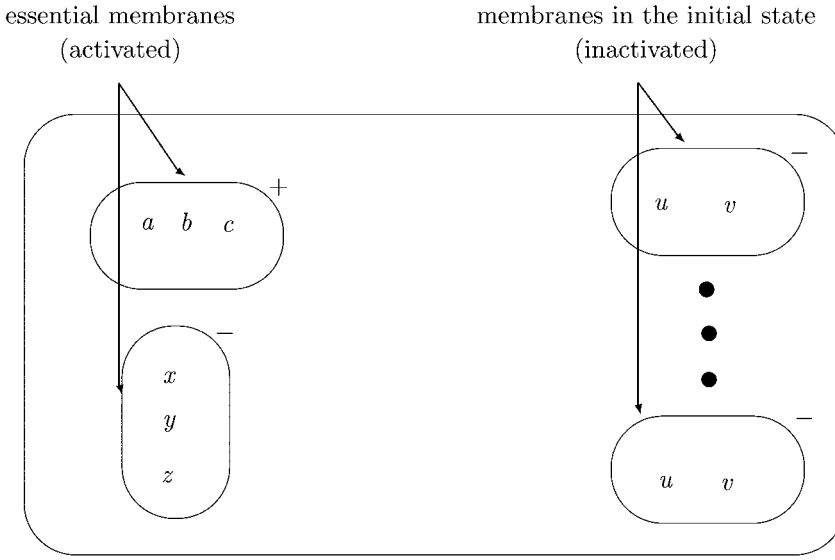



**Fig. 2.** A configuration of a self-activated P system

The evolution rules that can be applied to the multisets of objects present in the regions delimited by membranes can be of two types: *developmental rules*, in which the objects of a multiset placed inside a membrane evolve into a new multiset of objects, and *developmental/communicational rules*, in which the objects of a multiset placed inside a membrane can interact with the objects of another multiset placed in another membrane, so that both a modification and a displacement of the involved objects (between the two membranes) could be possible. It must be noticed that these systems are cooperative, i.e., the rules which are used can have a radius (which is the number of the objects involved in that rule) bigger than 1.

Due to the fact that inside the skin membrane we have a large number of membranes, essential or in the initial state (that is, active or inactive), in the representation of the membrane structure of those systems we use the following convention: if a pair of square brackets is placed inside a pair of braces, then it

means that the membrane which has been described in this way can be found in the system in an arbitrarily large number of copies. Such a notation will be used when describing the membranes in the initial state placed inside the system. For example, the membrane structure of the P system described in Figure 2 can be represented by the following expression of parentheses:

$$[_e [_1 \ ]_1 [_2 \ ]_2 \ \{ \ [_f \ ]_f \} ]_e .$$

We now give the definition of our systems:
*A self-activating P system* is a construct

$$\Pi = (V, T, \eta, \mu, W, R_e, R_b),$$

where:

(i) $V$ is an alphabet (the *total alphabet* of the system);
(ii) $T \subseteq V$(the *terminal alphabet*);
(iii) $\eta$ represents a description of the *initial state* of an inactive membrane (structure, objects and polarization);
(iv) $\mu$ is a *membrane structure*, consisting of a *skin membrane* (labelled here with $e$), $m$ *essential membranes* labelled (in a one-to-one manner) with elements of a set $\Lambda$ (here we will use the labels $1, 2, \ldots, m$) and, between braces, a membrane labelled with $f$ which is in the *initial state* (the one described by $\eta$), and which will appear in the system in an arbitrarily large number of copies.
(v) $W$ is a set with $m$ elements, which consists of strings over $V$,representing the multisets of objects placed in the $m$ regions of $\mu$ (in this set we do not describe the multisets of objects placed inside the inactive membranes, that is, the membranes in the initial form, because this is done in $\eta$).
(vi) $R_e$ is the set of rules applicable to objects placed in the region delimited by the skin membrane; all these rules are of the form
$r_e : [_0 a]_0^{\alpha_1} \to a[_0 \ ]_0^{\alpha_2}$,
for $\alpha_1, \alpha_2 \in \{+, -, 0\}$ and $a \in T$
(these rules are meant to sent terminal objects out of the system);
(vii) $R_b$ is a finite set of rules which are applicable to the objects placed inside all regions of the system, except the one delimited by the skin membrane. The rules from $R_b$ can be of the following forms:
(a) $[a]^{\alpha_1} \to [b]^{\alpha_2}$,
for $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $a, b \in V^*$
(object evolution rule, depending on the charge of the membrane, which can modify both the objects and the charge of the membrane involved).
(b) $[a]^{\alpha} + [b]^{\beta} \to [c]^{\gamma}[d]^{\gamma}$,
for $a, b, c, d \in V^*$, $\alpha, \beta, \gamma \in \{+, -, 0\}$
(membrane interconnecting rule, under the influence of the objects and of the charges of the two membranes; after connecting, the membranes have the same charge, and the object under which the connection was realized may change).

(c) $[a]^{\gamma}[b]^{\gamma} \to [c]^{\delta}[d]^{\delta}$,
for $a, b, c, d \in V^*$, $\gamma, \delta \in \{+, -, 0\}$
(communication/evolution rule, applicable to objects placed in the two interconnected membranes, which can alter both the objects involved and the common charge of the two membranes; this rule, also makes it possible to transfer objects between the two interconnected membranes).

(d) $[a]^{\gamma}[b]^{\gamma} \to [c]^{\alpha} + [d]^{\beta}$,
for $a, b, c, d \in V^*$, $\alpha, \beta, \gamma \in \{+, -, 0\}$
(membrane disconnecting rule, under the influence of the objects and of the common charge of the two membranes; after disconnecting, both the objects involved and the two charges can be altered).

(e) $[a]^{\alpha} + [b]^{\beta} \to [c]^{\gamma} + [d]^{\delta}$,
for $a, b, c, d \in V^*$, $\alpha, \beta, \gamma, \delta \in \{+, -, 0\}$
(communication/evolution rule, applicable to objects placed in two non-connected membranes, which can alter both the objects involved and the charge of the two membranes).

(f) $[a] \to a[\ ]$,
for $a \in T$ (communication rules).

The rules from $R$ are applied according to the following *principles*:

1. The rules are used as usual in the framework of membrane computing, that is, in the maximal parallel manner: in a step, any object which can evolve by a rule of any form, should evolve.
2. Each copy of an object, as well as each copy of a membrane, can be used only by one rule at a time, with the exception of the rules of type (a), (f) and of those of type (c), where only the objects involved in the rule are considered, and not the containing membranes. However, rules of type (c) cannot be applied at the same time with rules of type (b) or (d), due to the fact that these two types of rules imply an interconnection, respectively a disconnection between two membranes, while a rule of type (c) is a communication/evolution rule.
3. If more than one rule can be applied to the same objects in one membrane, then the rule which is applied is non-deterministically chosen (there is no priority relation among rules).
4. All objects and membranes not specified in a rule and which do not evolve are moved unchanged to the next step. For example, if applying the following rule of type (c):

$$[a_1, b_1][c_1, d_1] \to [b_2][a_2, c_2], \quad a_1, b_1, c_1, d_1, a_2, b_2, c_2 \in V,$$

to the two interconnected membranes

$$[x, y, a_1, b_1][z, c_1, d_1], \quad x, y, z \in V,$$

we obtain the following configuration:

$$[x, y, b_2][z, c_2, a_2].$$

The electrical charge of a membrane at one step is given by all the rules which have been applied inside that membrane at the previous step, in a cumulative manner.

The concepts of *configuration at a given time, initial configuration, transition among configurations, computation with respect to $\Pi$, complete computation* and *$L(\Pi)$* as *the language generated by $\Pi$*, are defined as usual in the framework of the P systems theory, as shown in [6].

A distinction must be made between the rules of type (c) and those of type (e), though it seems that both rules have the same consequence: the transfer (and maybe the modification) of objects between two membranes. While the rules of type (c) can be applied only between interconnected membranes, rules of type (e) do not require interconnection. Also, while rules of type (e) can be applied only once to a pair of membranes (during a step), those of type (c) can be applied several times to the same two membranes (in the same step). The purpose of the type (c) of rules is to transfer as many objects as possible (in a bound manner), between two interconnected membranes, at the same time (a bound manner means the following: although we cannot be sure in which membrane we have transferred the objects, because the membrane is non-deterministically chosen, we can be sure that all the objects involved will be transferred in that particular membrane).

We define the *diameter* of a rule $r \in R_b$, denoted by $diam(r)$, as the maximal cardinality od a multiset of objects which appears in the rule $r$. Then, the diameter of a system $\Pi$ is $diam(\Pi) = max\{diam(r) \mid r \in R_b\}$.

### 2.1 Example

We illustrate the above definitions with the following example: we consider a self-activating P system of diameter 2, which given a number, will return whether or not this number is prime. The P system is of the following form:

$$\Pi = (V, T, \eta, \mu, W, R_e, R_b),$$

where:

1. $V = \{a, b, c, u, v, x, y, y', c_p, d_p, a_p, \mathtt{no}, e, q\}$,
2. $T = \{e, \mathtt{no}\}$,
3. $\eta = \left[_f \; a_p\right]_f$,
4. $\mu = \left[_e \left[_1 \, \right]_1 \left\{ \, \left[_f \, \right]_f \right\} \right]_e$,
5. $W = \{a^n b^n c^n v\}$,
6. $R_e = \{r_e : [_e a]_e \to a[_e \, ]_e \mid a \in T \}$,
7. $R_b$ contains the following rules:
   (a) $[a^2] \to [e]$;
   (b) $[e] \to e[\,]$;
   (c) $[b^2] \to [d]$;
   (d) $[d\, v] \to [c_p]$;

(e) $[c_p] + [a_p] \to [u][y\, y\, q]$;
(f) $[c][\,] \to [c'][x]$;
(g) $[d][\,] \to [d'][y]$;
(h) $[c'] \to [c]$;
(i) $[d'] \to [d]$;
(j) $[u] \to [d_p]$;
(k) $[d_p][\,] \to [v] + [\,]$;
(l) $[x\, y] \to [y']$;
(m) $[x\, y'] \to [y]$;
(n) $[q\, y\, y'] \to [\mathtt{no}]$;
(o) $[\mathtt{no}]^0 \to \mathtt{no}[\,]^{-1}$.

The system evolves as follows:

**Step 1:** a number of $\lfloor \frac{n}{2} \rfloor$ copies of the objects $e$ and $d$ appear in the system (due to the application of rules 1 and 3).

**Step 2:** all (terminal) objects $e$ are sent outside the original membrane, to the skin membrane (by the use of rule 2);
the object $c_p$ appears (by the use of rule 4). (This means that the membrane is prepared for an interconnection with another membrane, which is in the initial state. The object $c_p$ is a special object, being designated only for interconnecting membranes.) It must be stressed out that due to the rule number 4, at each interconnection, one copy of the object $d$ will be lost; though, by the end of the computation, we will have a total of $\lfloor \frac{n}{2} \rfloor$ interconnections.

**Step 3:** all the $e$ objects get out of the system (by the use of rule $r_e$), and are counted;
the membranes are interconnecting (one membrane in the initial state is activated) (by the use of rule 5) and one copy of the object $u$, two copies of the object $y$, and one copy of the object $q$ are sent to the first respectively to the second membrane.

**Step 4:** all copies of the object $c$ ($n$ copies) are doubling: half of them ($n$ copies) are transformed into $x$, and sent to the next membrane, and half ($n$ copies) are marked with $'$ and maintain their position inside the membrane (by the use of rule 6);
the same happens with the copies of the object $d$ (by the use of rule 7);
the object $u$ is transforming into $dp$ (by the use of rule 10)(This means that the membrane is prepared for disconnecting).

**Step 5:** all the objects $c'$ and $d'$ turn back to $c$ and respectively $d$ (by the use of rules 8 and 9);
the membranes disconnect, and from this moment on they have the following evolution:

The first membrane evolves as it did on steps 2 to 5 until no copies of the object $d$ will exist (that means $\lfloor \frac{n}{2} \rfloor - 1$ times); during these steps, the membrane

interconnects with a membrane in the initial state, and $n$ copies of the object $x$ as well as a decreasing number of copies of the objects $y$ are sent inside this membrane.

In parallel, in the second membrane is checked whether or not $n$ (which is the number of copies of the object $x$) divides by $k$, where $k$ is a smaller number (the number of copies of the object $y$). It has to be mentioned that at each interconnection, the number of copies of the object $y$ decreases (by one), from $\lfloor \frac{n}{2} \rfloor + 1$ to 2. If $k$ does not divide $n$, the (terminal) object `no` appears (by the use of rule 14), and at the next step it is sent out of this membrane (by the use of rule 15), and then out of the system (by rule $r_e$), and counted.

Therefore, if by the time we reach a complete configuration outside the system we have counted an equal number of copies of the objects $e$ and `no` (this means that $n$ does not divide any number from $\lfloor \frac{n}{2} \rfloor + 1$ to 2), then the number $n$ is prime.

*Remark 1.* The interconnection of membranes, as well as the transfer of objects from one membrane to the other, was made by the help of rules of type (b), (c) and (d) (this means that the process has been implemented in more steps), and not by rules of type (e) (that is, in a single step), because, if we have used rules of type (e) we could not have been sure that all copies of the object $x$ (which have appeared from the objects $c$ due to rule 6) as well as all copies of the object $y$ (which have appeared from the objects $d$ due to rule 7) would have been sent to the same membrane.

## 3 Solving SAT in Linear Time

The main reason for considering the self-activation of membranes is to provide an enhanced parallelism, in order to solve complex problems in a feasible time. In order to prove the use of our systems we consider the case of the SAT problem.

**Theorem 1.** *The SAT problem can be solved by a self-activating P system of diameter 2, in a time which is linear in the number of variables and the number of clauses.*

*Proof.* Let us consider $n$ variables $x_1, ..., x_n, n \geq 1$, and a propositional formula

$$\gamma = C_1 \wedge C_2 \wedge ... \wedge C_m,$$

with

$$C_i = y_{i,1} \vee ... \vee y_{i,p_i},$$

for some $m \geq 1, p_i \geq 1$, and $y_{i,j} \in \{x_k, \neg x_k \mid 1 \leq k \leq n\}$, for each $1 \leq i \leq m$ and $1 \leq j \leq p_i$.

We construct the P system:

$$\Pi = (V, T, \eta, \mu, W, R_e, R_b)$$

with the components

$$V = \{X_1, X_2, \ldots, X_n\} \cup \{X'_1, X'_2, \ldots, X'_n\}$$
$$\cup \{T_1, F_1, \ldots, T_n, F_n\} \cup \{T'_1, F'_1, \ldots, T'_n, F'_n\} \cup \{T''_1, F''_1, \ldots, T''_n, F''_n\}$$
$$\cup \{CP, AP, DP, \overline{CP}\} \cup \{Q, Q_0, Q_1, \ldots, Q_n\} \cup \{O, V_0, \sharp\}$$
$$\cup \{CP_1, CP'_1, CP_2, CP'_2, \ldots, CP_m, CP'_m\} \cup \{S_1, S_2, S_3, S_4\},$$

$$T = \{CP_{m+1}\},$$
$$\eta = [_f AP]^-_f,$$
$$\mu = \left[_e \, [_1 \, ]_1 \, [_2 \, ]_2 \, \cdots [_m \, ]_m \, [_{m+1} \, ]_{m+1} \, \left\{ \, [_f \, ]_f \right\} \right]_e,$$
$$W = \{w_i \mid 1 \le i \le m+1\},$$

where $w_i = \{a \mid a \in C_i\} \cup \{R^n, O, CP'_i\}$, for all $i = 1, \ldots, m$, and $w_{m+1} = \{X_j \mid j = 1, \ldots, n\} \cup \{V_0\}$,

$$R_e = \{r_e : [_e a]^0_e \to a[_e \, ]^+_e, \; a \in T \; \},$$

while the set $R_b$ contains the following rules ($i = 1, \ldots, n$, and $l = 1, \ldots, m$):

(1) $[V_0, X_i] \to [X_i, \overline{CP}]$;
(2) $[\overline{CP}]^0 \to [CP]^+$;
(3) $[CP, X_i]^+ + [AP]^- \to [S_1, T''_i][S_2, F''_i]$;
(4) $[S_1][S_2] \to [V_0, S_3, DP][V_0, S_3]$;
(5) $[X_i][\,] \to [X'_i][X'_i]$;
(6) $[T_i][\,] \to [T'_i][T'_i]$;
(7) $[F_i][\,] \to [F'_i][F'_i]$;
(8) $[T''_i] \to [T'_i]$;
(9) $[F''_i] \to [F'_i]$;
(10) $[S_3] \to [Q]$;
(11) $[DP][\,] \to [\,][\,]$;
(12) $[T'_i] \to [T_i]$;
(13) $[F'_i] \to [F_i]$;
(14) $[X'_i] \to [X_i]$;
(15) $[Q] \to [S_4]$;
(16) $[S_4, V_0] \to [CP_1]$;
(17) $[S_4]^+ \to [\sharp]^0$;
(18) $[O] \to [\sharp]^+$;
(19) $[CP'_l, R]^+ + [AP]^- \to [Q_l][Q_l]$;
(20) $[Q_l][Q_l] \to [CP'_l, O, DP][CP'_l, O]$;
(21) $[R][\,] \to [R'][R']$;
(22) $[R'] \to [R]$;
(23) $[CP_l, T_i] + [CP'_l, T_i]^+ \to [CP_{l+1}, T_i] + [CP'_l, T_i]^+$;
(24) $[CP_l, F_i] + [CP'_l, F_i]^+ \to [CP_{l+1}, F_i] + [CP'_l, F_i]^+$;
(25) $[CP_{m+1}] \to CP_{m+1}[\,]$.

The systems evolves as follows:

By using rules (1) to (15) and rule (17), in 5 steps, all the objects from one membrane which would contain at least one unexpanded variable together with all the truth assignments of the rest of the variables (which have been expanded until that step) are copied into a membrane in the initial state in the following manner:

- in the first step the object $V_0$ is transformed into $\overline{CP}$ (using rule (1)), if and only if at least one copy of an unexpanded variable still exists inside the membrane;
- in the second step, the membrane is prepared for an interconnection with an initial state membrane (applying rule (2) the object $CP$ appears inside the membrane, and its polarization becomes positive);
- in the third step the interconnection takes place (rule (3)) and one of the variables is expanded into its truth assignments, $truth$ ($T''$) into the first membrane, and $false$ ($F''$) into the second;
- in the fourth step (using rules (5), (6), (7), (8), and (9)) all the objects representing unexpanded variables or truth assignments of some variables are duplicated into the second membrane and at the same time are marked with $'$ (this is done in such a way that the objects will not be duplicated more that once); also (by applying rule (4)), the object $V_0$ appears in both membranes, while the object $DP$ appears only in one membrane (the first one), this implies that the membranes will be disconnected at the next step;
- in the fifth step the membranes are disconnected (using rule (11)), and all the objects which have been marked with $'$ are turned back to the initial form (by applying rules (12), (13), and (14)).

In the next step, if an unexpanded variable still exists inside the membrane, the process is repeated. Otherwise, the object $V_0$ does not disappear from the membrane, and at the next step, together with the object $S_4$ it will be transformed into $CP_1$ (rule (16)). It must be observed that the object $S_4$ appears during each cycle in the sixth step, and if object $V_0$ does not disappear in the same step (that is, if there are no unexpanded variables inside the membrane), rule (16) is applied. But, if the $V_0$ object disappears in the sixth step, (there still exists at least one unexpanded variable in the membrane), object $S_4$ is transformed in the third step of the next cycle into $\sharp$, object which will never evolve.

Therefore, after $5n + 2$ steps, from one membrane containing all the $n$ variables we will obtain $2^n$ membranes containing all the possible combinations of all the truth assignments of the variables.

In the same way, one of the membranes containing the truth assignments which would validate one of the $m$-th clauses (in the initial configuration we have $m$ such membranes, one for each clause) will be copied in $2^n$ identical membranes, using the rules (6), (7), (12), (13), and rules (18) to (22). Here, we control the number of duplications in the following way: at each time such a membrane is copied in an initial state membrane, one copy of the object $R$ disappears. Due to the fact that in the initial configuration in each such membrane we have $n$

copies of the object $R$, one may conclude that the number of identical copies obtained from one such membrane will be $2^n$. It must be also mentioned that the cycle of membrane duplication takes three steps: first the membrane is polarized (positively), second the membrane interconnects with an initial state membrane, and third all the objects are copied, and the membrane is duplicated. After that, if at list one copy of the object $R$ still exists, the cycle will be repeated.

In conclusion, in $5n + 2$ steps, inside the region delimited by the skin membrane we will have $(m + 1) * 2^n$ membranes. From here, in a number of steps linear with the number of clauses ($m$ steps), we will check if one of the combination of all the truth assignments of the variables satisfies all the clauses (this is done using rules (23) and (24)). If so, the object $CP_{m+1}$ appears inside this membrane, and at the next step, applying rule (25), it is sent to the region delimited by the skin membrane. From here, due to the fact that the object $CP_{m+1}$ is in the terminal alphabet, it is eliminated from the system by applying the rule $r_e$.

In conclusion, in $5n + m + 4$ steps at least one copy of the object $CP_{m+1}$ is sent out of the system, if and only if the propositional formula $\gamma$ can be satisfied. If the formula $\gamma$ cannot be satisfied, then the system will stop before reaching this step. □

## 4 Final Remarks

Until now, the strategies of creating an exponential space (in a linear or polynomial time) in the framework of membrane computing have considered only the case when new "material" appears inside the system, such as new membranes (by duplication or creation) or new string-objects. Here, we consider another strategy: the exponential space already exists, in the form of an unused arbitrarily large number of identical membranes, and our problem is only how to activate this space. Using such a system, we have proved (by solving the SAT problem) that computationally hard problems can be approached in this new framework.

Although very likely to be obtained (due to the cooperative rules) the computational universality of these systems was not discussed, and it remains as an open problem.

A problem of (mathematical) interest about self-activating P systems is also to consider the case when the base cell has a more complex membrane structure (so new rules applicable only in those regions will appear), and the case when more types of base cell (that is, different membrane structures, and different rules associated with regions) can be found inside the skin membrane. In this latter case, we are going into a new field, that of the interaction between P systems and of the networks of P systems.

## References

1. J. Castellanos, A. Rodriguez-Paton, Gh. Păun, Computing with membranes: P systems with worm-objects, IEEE 7th. Intern. Conf. on String Processing and Information Retrieval, SPIRE 200, La Coruna, Spain, 64–74.
2. R. Freund, C. Martin-Vide, Gh. Păun, Computing with membranes: Three more collapsing hierarchies, submitted 2000.
3. S.N. Krishna, R. Rama, P systems with replicated rewriting, Journal of Automata, Languages and Combinatorics, 6, 3 (2001), 345–350.
4. C. Martin-Vide, Gh. Păun, A. Rodriguez-Paton, On P systems with membrane creation, Computer Science Journal of Moldova, 9, 2 (2001), 134–145.
5. A. Păun, On P systems with membrane division, in vol. Unconventional Models of Computation (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer-Verlag, London, 2000, 187–201.
6. Gh. Păun, Computing with membranes, Journal of Computer and System Science, 61, 1 (2000), 108–143.
7. Gh. Păun, P systems with active membranes: Attacking NP complete problems, Journal of Automata, Languages and Combinatorics, 6, 1 (2001), 75–90.

# Energy–Controlled P Systems

Rudolf Freund

Department of Computer Science
Technische Universität Wien, Wien, Austria
rudi@emcc.at

**Abstract.** As already considered in [13], we investigate P systems where each evolution rule "produces" or "consumes" some quantity of energy, in amounts which are expressed as integer numbers. Yet in contrast to P systems with energy accounting as considered in [13], for energy-controlled P systems we demand that in each evolution step and in each membrane the total energy consumed by the application of a multiset of evolution rules has to be the maximum possible within a specific non-negative range. Only equipped with this control feature, energy-controlled P systems are very powerful. In the case of multisets of symbol objects we find that energy-controlled P systems with even only one membrane and an energy range of $\{0,1\}$ for the total energy involved in an evolution step characterize the recursively enumerable sets of vectors of natural numbers (without using catalysts or priorities or membrane dissolving features). In the case of string objects similar results can be obtained. Energy-controlled P systems with even only one membrane and the minimal energy range of $\{0\}$ for the total energy involved in an evolution step at least generate any set of vectors of natural numbers that can be generated by matrix grammars without appearance checking.

## 1 Introduction

Membrane systems were introduced by Gheorghe Păun (and therefore then called P systems) in [8]; they are a class of distributed parallel computing models, inspired from the way how alive cells process chemical compounds, energy, and information. The basic part of a P system is a *membrane structure* consisting of several membranes placed within one unique surrounding membrane, the so-called skin membrane. All the membranes can be labelled in a one-to-one manner by natural numbers where we always label the outermost membrane (skin membrane) with 1. In that way, a membrane structure can uniquely be described by a string of correctly matching parentheses, where each pair corresponds to a membrane. For example, the membrane structure depicted in Figure 1, which within the skin membrane contains two inner membranes labelled by 2 and 3, etc., is described by $[_1[_2]_2[_3[_4]_4[_5[_7]_7]_5[_6]_6]_3]_1$.

In a *region* $k$, which is the volume delimited by membrane $k$ and its inner membranes, *multisets* of *objects* evolve according to *evolution rules* associated with the region; a *computation* consists of transitions among system *configurations*; the *result* of a halting computation is the vector of the multiplicities of

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 247–260, 2003.

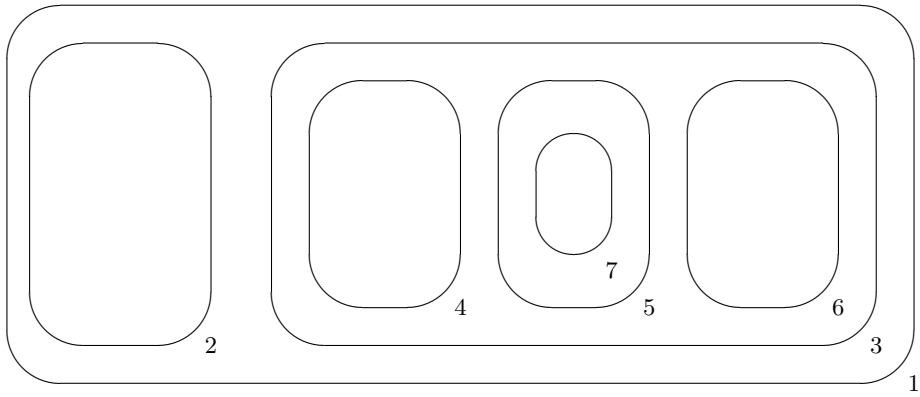



**Fig. 1.** Membrane structure $[_1[_2]_2[_3[_4]_4[_5[_7]_7]_5[_6]_6]_3]_1$.

objects present in the final configuration in a specified *output membrane* or of objects which leave the external membrane of the system (the *skin* membrane) during a computation. In that way, a P system *computes* a set of vectors of natural numbers. Many variants characterize the family of recursively enumerable sets of vectors of natural numbers (which are exactly the Parikh sets associated with recursively enumerable languages). Strings over a given alphabet can be considered as the objects in a P system themselves. Using adequate string processing operations (rewriting, splicing, etc.), for various classes of P systems also characterizations of recursively enumerable languages are obtained; details can be found in [1], [3], [4], [8], [9], [10], etc. (see the P Systems Web Page [7]).

Considering the energy balancing of processes in a cell was formulated as an explicit research topic in [11] and first investigated in [13]. There the energies of all rules to be used in a given step in a membrane are summed up; if the total amount of energies is positive, then this multiset of rules can be applied if it is maximal with this property (no further rule can be applied to the remaining objects in the membrane such that the total energy is still positive), otherwise it cannot be applied. Moreover, the energy passes from one step to the next one (initially, we assume that there is no energy in the system). If the energy accumulated at a given step in a membrane is larger than a given threshold associated with that membrane, then the membrane is dissolved and the energy accumulated in this membrane is consumed (as usual in P systems, the objects from this region pass to the surrounding region, whereas the evolution rules get lost).

The energy-controlled P systems proposed in this paper handle the energy amounts in a different way: The energies of all rules to be used in a given step in a membrane are summed up; if the total energy is within a given range, then this multiset of rules can be applied if it is maximal with this property (no other multiset of rules can be applied to the present objects in the membrane such that the total energy is larger but still within the desired range, and no

extension of the multiset is possible such that these properties are preserved); otherwise it cannot be applied. These features for controlling the total amount of energy consumed by applying a multiset of rules within a membrane are very powerful, i.e., universality of energy-controlled P systems can already be obtained with the simplest membrane structure $[_1]_1$ and an energy range of $\{0, 1\}$ (when considering only symbols as objects as well as when considering strings as objects), even without using catalysts or priorities on the rules (as we only need the skin membrane, membrane dissolving features can be omitted, too). Moreover, energy-controlled P systems with only one membrane and the minimal energy range of $\{0\}$ for the total energy involved in an evolution step at least generate any set of vectors of natural numbers that can be generated by matrix grammars without appearance checking.

## 2 Prerequisites

The reader is assumed to be familiar with the basic notions of formal language theory, e.g., see [2] and [14]. We only give the following definitions:

For an alphabet $V$, by $V^*$ we denote the free monoid generated by $V$ under the operation of concatenation; the *empty word* is denoted by $\lambda$, and $V^+ := V^* \setminus \{\lambda\}$. For any word $w \in V^*$ and any $X \in V$, $|w|_X$ represents the number of occurrences of the symbol $X$ in $w$. For $w \in V^*$, $V = \{a_1, \ldots, a_n\}$, by $\Psi_V(w)$ we denote the Parikh vector of $w$, i.e., $\Psi_V(w) = (|w|_{a_1}, \ldots, |w|_{a_n})$; this is extended to languages in the natural way. By $RE$ and $CF$ we denote the family of recursively enumerable languages and the family of context-free languages, respectively. For a family $F$ of languages, by $PsF$ we denote the family of Parikh sets of vectors associated with languages in $F$. As in [2], we consider two string languages $L$, $L'$ to be equal if $L \setminus \{\lambda\} = L' \setminus \{\lambda\}$.

A multiset over an alphabet $V$ is represented by a string over $V$ and by all its permutations, and conversely, each string precisely identifies one multiset; the Parikh vector associated with a string indicates the multiplicities of each element of $V$ in the corresponding multiset. Thus, when speaking of a "multiset" $w \in V^*$ we understand the multiset identified by $w$.

A *matrix grammar* is a construct $G = (N, T, (M, F), S)$ where $N$ and $T$ are sets of *non-terminal* and *terminal symbols,* respectively, with $N \cap T = \emptyset$, $S \in N$ is the start symbol, $M$ is a finite set of matrices, $M = \{m_i \mid 1 \leq i \leq n\}$, where the matrices $m_i$ are sequences of the form $m_i = (m_{i,1}, \ldots, m_{i,n_i})$, $n_i \geq 1$, $1 \leq i \leq n$, and the $m_{i,j}$, $1 \leq j \leq n_i$, $1 \leq i \leq n$, are context-free productions over $(N, T)$, and $F$ is a subset of $\bigcup_{1 \leq i \leq n,\, 1 \leq j \leq n_i} \{m_{i,j}\}$.

For $m_i = (m_{i,1}, \ldots, m_{i,n_i})$ and $v, w \in (N \cup T)^*$ we define $v \Longrightarrow_{m_i} w$ if and only if there are $w_0, w_1, \ldots, w_{n_i} \in (N \cup T)^*$ such that $w_0 = v$, $w_{n_i} = w$, and for each $j, 1 \leq j \leq n_i$,

- **either** $w_j$ is the result of the application of $m_{i,j}$ to $w_{j-1}$,
- **or** $m_{i,j}$ is not applicable to $w_{j-1}$, $w_j = w_{j-1}$, and $m_{i,j} \in F$.

The language generated by $G$ is

$$L(G) = \{w \in T^* \mid S \Longrightarrow_{m_{i_1}} w_1 \ldots \Longrightarrow_{m_{i_k}} w_k,\ w_k = w, \\ w_j \in (N \cup T)^*,\ m_{i_j} \in M \ \text{ for } 1 \leq j \leq k, k \geq 1\}.$$

A non-terminal symbol $A \in N$ is said to be used in the appearance checking mode, if there exists at least one production of the form $A \to \alpha$, $\alpha \in (N \cup T)^*$, that appears in $F$.

In [2] the following normal form for matrix grammars was established:

A matrix grammar $G = (N, T, (M, F), S)$ is said to be in the *binary normal form* if $N = N_1 \cup N_2 \cup \{S, \#\}$, with these three sets being mutually disjoint, and the matrices in $M$ are of one of the following forms:

1. $(S \to XA)$, with $X \in N_1$, $A \in N_2$,
2. $(X \to Y, A \to \alpha)$, with $X, Y \in N_1$, $A \in N_2$, $\alpha \in (N_2 \cup T)^*$,
3. $(X \to Y, A \to \#)$, with $X, Y \in N_1$, $A \in N_2$,
4. $(X \to \lambda, A \to \alpha)$, with $X \in N_1$, $A \in N_2$, and $\alpha \in T^*$.

Moreover, there is only one matrix of type 1 and $F$ consists exactly of all rules $A \to \#$ appearing in matrices of type 3; # is a trap symbol; once introduced, it is never removed. A matrix of type 4 is used only once, in the last step of a derivation.

According to Lemma 1.3.7 in [2], for each matrix grammar $G$ an equivalent matrix grammar $G'$ in the binary normal form can be constructed.

For an arbitrary matrix grammar $G = (N, T, (M, F), S)$, let us denote by $ac(G)$ the cardinality of the set $\{A \in N \mid A \to \alpha \in F\}$. From the construction in the proof of Lemma 1.3.7 in [2] one can see that if we start from a matrix grammar $G$ and we get the grammar $G'$ in the binary normal form, then $ac(G') = ac(G)$.

A matrix grammar is called a *matrix grammar without appearance checking*, if $ac(G) = 0$ (i.e., $F = \emptyset$).

In [5], an even stronger normal form was proved:

A matrix grammar $G = (N, T, (M, F), S)$ is said to be in the *strong binary normal form,* if $N = N_1 \cup N_2 \cup \{S, \#\}$, with these three sets being mutually disjoint, and the matrices in $M$ are of one of the following forms:

1. $(S \to XA)$, with $X \in N_1$, $A \in N_2$,
2. $(X \to Y, A \to \alpha)$, with $X, Y \in N_1$, $A \in N_2$, $\alpha \in (N_2 \cup T)^*$,
3. $(X \to Y, A \to \#)$, with $X, Y \in N_1$, $A \in N_2$,
4. $(X \to \lambda)$, with $X \in N_1$.

Moreover, there is only one matrix of type 1 and only one of type 4, and $F$ consists exactly of all rules $A \to \#$ appearing in matrices of type 3, where # is the trap symbol. The matrix of type 4 is used only once, in the last step of a derivation (and the non-terminal in this terminal matrix never occurs on the left-hand side of a production in the other matrices). Finally, as the most important feature, we have $ac(G) \leq 2$.

The family of languages generated by matrix grammars is denoted by $MAT_{ac}$; the family of languages generated by matrix grammars without appearance checking is denoted by $MAT$. It is known that $CF \subset MAT \subset MAT_{ac} = RE$. Further details about matrix grammars can be found in [2] and in [14]. Due to the results shown in [6], we have $\left\{a^{2^n} \mid n \geq 1\right\} \in MAT_{ac} \setminus MAT$.

## 3 Energy-Controlled P Systems with Symbols

An *energy-controlled P system (ECP system for short)* (of degree $m$, $m \geq 1$) is a construct

$$\Pi = (V, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, d_1, \ldots, d_m),$$

where:

1. $V$ is an alphabet; its elements are called *objects* or *symbols*;
2. $\mu$ is a membrane structure consisting of $m$ membranes, with the membranes and the regions labelled in a one-to-one manner with $1, 2, \ldots, m$; the skin membrane is labelled with 1;
3. $w_1, \ldots, w_m$ are multisets over $V$ associated with the regions $1, 2, \ldots, m$ of $\mu$;
4. $R_i, 1 \leq i \leq m$, are finite sets of *evolution rules* over $V$. These rules are of the forms $a \to v \langle e \rangle$, where $a \in V$, $v$ is a string over $V \times \{here, out, in\}$, and $e$ is an integer number which specifies the *energy* associated with the rule;
5. $d_1, \ldots, d_m$ are non-negative numbers indicating the *maximal total energy* allowed for an evolution step in each membrane.

When presenting the evolution rules, in general the indication "here" is omitted. The membrane structure and the multisets in $w_1, \ldots, w_m$ constitute the *initial configuration* of the system. The application of a rule $a \to v \langle e \rangle$ in a region containing a multiset $w$ means to remove a copy of the object $a$ from $w$ and to add the objects specified by $v$, following the prescriptions given by $v$: If an object in $v$ appears in the form $(b, here)$, then it remains in the same region; if it appears in the form $(b, out)$, then a copy of the object $b$ will be introduced in the region which surrounds the region of the rule $a \to v \langle e \rangle$; if it appears in the form $(b, in)$, then a copy of $b$ is introduced in one of the regions of the membranes placed directly inside the region of the rule $a \to v \langle e \rangle$, non-deterministically chosen, if such a region exists, otherwise the rule cannot be applied.

In each step and in each membrane, the rules are used according to the following condition controlling the energy balance of the rules used in each membrane: The energies of all rules to be used in a given step in membrane $i$ are summed up; if the total is in the closed interval $[0, d_i]$, then this multiset of rules can be applied if it is maximal with this property (no other multiset of rules can be applied to the objects in membrane $i$ such that the total energy is larger but still within the desired range 0 to $d_i$, and no extension of the multiset is possible such that these properties are preserved); otherwise it cannot be applied.

By using the rules of $\Pi$ in the way described above, we can pass from a configuration of the system to another configuration. A sequence of such transitions

between configurations is called a *computation* of $\Pi$. A computation is *successful* if and only if it halts, i.e., in the last configuration there is no membrane where a multiset of rules is applicable to the objects present in the membrane such that the energy-controlling condition is fulfilled. The result of a successful computation is $\Psi_V(w)$, where $w$ describes the multiset of objects from $V$ being in the skin membrane at the end of a halting computation; we say that this vector is *generated* by $\Pi$. By $N_1(\Pi)$ we denote the set of all vectors generated by $\Pi$ in that way. Another variant to get the result of a successful computation is to consider the multiset $w$ of objects from $V$ which have left the skin membrane during the computation. By $N_0(\Pi)$ we denote the set of all vectors $\Psi_V(w)$ generated by $\Pi$ in that way.

Finally, energy-controlled P systems as defined above can also be considered as generating devices for string languages: The symbols sent out during a successful computation in the sequence they are sent out can be interpreted as a string result of the computation. If several symbols are sent out at the same time, any permutation of these symbols can be taken (e.g., see [12]). The string language generated by $\Pi$ is denoted by $N_2(\Pi)$.

The family of all sets $N_i(\Pi)$, $i \in \{0,1,2\}$, generated by systems of degree at most $m$, $m \geq 1$, and with the maximal energy bound associated with a membrane at most $d$ is denoted by $ECP_{i,m}(d)$; if no bound on the number of membranes or on the maximal energy associated with each region is imposed, then we replace the corresponding parameters by $*$.

We first prove that energy-controlled P systems can simulate matrix grammars even with the simplest membrane structure and only two energy values 0 and 1.

**Theorem 1.** *$PsRE = ECP_{1,m}(1)$, for all $m \geq 1$.*

*Proof.* Due to the equality $RE = MAT_{ac}$, more exactly, to the equality $PsRE = PsMAT_{ac}$, we only have to prove the inclusion $PsMAT_{ac} \subseteq ECP_{1,1}(1)$.

Let $G = (N, T, (M, F), S)$ be a matrix grammar in the strong binary normal form, with $N = N_1 \cup N_2 \cup \{S, \#\}$ and matrices of the four forms mentioned above. Assume that we have $s$ matrices of the form $(X \to Y, A \to x)$, with $X, Y \in N_1$, $x \in (N_2 \cup T)^*$, and $t$ matrices of the form $(X \to Y, A \to \#)$, $X, Y \in N_1$, $A \in N_2$. Moreover, let $(f \to \lambda)$ be the unique terminal matrix. We label the matrices $(X \to Y, A \to x)$ by $m_i$, $1 \leq i \leq s$, and the matrices of the form $(X \to Y, A \to \#)$ by $m_{s+j}$, $1 \leq j \leq t$.

We construct the P system (of degree 1)

$$\Pi = (V, [\,]_1, w_1, R_1, 1),$$

with

$V = N \cup T \cup \{Z\} \cup \{Z_j \mid 1 \leq j \leq t\} \cup \{X_j \mid 1 \leq j \leq t\}$,

$w_1 = X_0 A_0 Z Z_1 \dots Z_t$, for $(S \to X_0 A_0)$ being the initial matrix of $G$,

and the sets of rules $R_1$ containing the following rules:

1. For each matrix $m_i : (X \to Y, A \to x) \in M$, $1 \leq i \leq s$, with $X, Y \in N_1$, $x \in (N_2 \cup T)^*$, we introduce the rules
   $X \to Y \ \langle -(3i + 1 + k) \rangle$,
   $A \to x \ \langle +(3i + 1 + k) \rangle$.
2. For each matrix $m_{s+j} : (X \to Y, A \to \#) \in M$, $1 \leq j \leq t$, $X, Y \in N_1$, $A \in N_2$, we introduce the rules
   $X \to Y \ \langle -(3(s + j) + 1 + k) \rangle$,
   $A \to \# \ \langle +(3(s + j) + 2 + k) \rangle$,
   $Z_j \to Z_j \ \langle +(3(s + j) + 1 + k) \rangle$.
3. We also introduce the following rules which are always applicable due to their zero energy balance:
   $Z \to Z \ \langle 0 \rangle$,
   $\alpha \to \alpha \ \langle 0 \rangle$, for all $\alpha \in N_2$.
4. The rule
   $\# \to \# \ \langle 0 \rangle$
   guarantees that a computation having introduced the trap symbol # will never stop.
5. For "cleaning up" the additional symbols $Z$ and $Z_j$, $1 \leq j \leq t$, we finally use the following rules:
   $f \to X_1 \ \langle -(3(s + t + 1) + 1 + k) \rangle$,
   $Z \to \lambda \ \langle +(3(s + t + 1) + 1 + k) \rangle$;
   $X_j \to X_{j+1} \ \langle -(3(s + t + j + 1) + 1 + k) \rangle$, $1 \leq j < t$,
   $Z_j \to \lambda \ \langle +(3(s + t + j + 1) + 1 + k) \rangle$, $1 \leq j < t$;
   $X_t \to \lambda \ \langle -(3(s + 2t + 1) + 1 + k) \rangle$,
   $Z_t \to \lambda \ \langle +(3(s + 2t + 1) + 1 + k) \rangle$.

The constant $k$ has to be defined in such a way that the sum of two energy values appearing in the rules above is greater than any of these values, hence, we take $k := 6(s + 2t + 3)$; this guarantees that when choosing a rule with a negative energy value only one of the rules with the corresponding positive value, eventually +1, can be chosen.

Assume that at some moment we have a multiset $XwZZ_1 \dots Z_t$; initially, $X = X_0$ and $w = A_0$, for $(S \to X_0 A_0)$ being the initial matrix of $G$.

As long as the additional symbol $Z$ will exist, the rule $Z \to Z \ \langle 0 \rangle$ can be used, hence the computation cannot stop; yet $Z$ can only be removed in combination with $f$; remember that $f$ - on the left-hand side of a production in $G$ - only appears in the terminal matrix $(f \to \lambda)$. As long as non-terminal symbols from $G$ are present, the computation cannot stop because of the rules $\alpha \to \alpha \ \langle 0 \rangle$, for all $\alpha \in N_2$. Therefore, a successful computation has to lead to a configuration with $fwZZ_1 \dots Z_t$ and $w \in T^*$; by using the rules

$f \to X_1 \ \langle -(3(s + t + 1) + 1 + k) \rangle$,
$Z \to \lambda \ \langle +(3(s + t + 1) + 1 + k) \rangle$;
$X_j \to X_{j+1} \ \langle -(3(s + t + j + 1) + 1 + k) \rangle$, $1 \leq j < t$,
$Z_j \to \lambda \ \langle +(3(s + t + j + 1) + 1 + k) \rangle$, $1 \leq j < t$;
$X_t \to \lambda \ \langle -(3(s + 2t + 1) + 1 + k) \rangle$,
$Z_t \to \lambda \ \langle +(3(s + 2t + 1) + 1 + k) \rangle$

in this sequence (here the energy balance is always 0) we finally get only $w$ in the skin membrane and the computation halts.

If we use a rule $X \to Y \ \langle -(3i+1+k) \rangle$ associated with a matrix $m_i : (X \to Y, A \to x)$, for some $1 \leq i \leq s$, then at the same step we also have to use the rule $A \to x \ \langle +(3i+1+k) \rangle$. Indeed, all the other rules, even those different from $Z \to Z \ \langle 0 \rangle$, $\alpha \to \alpha \ \langle 0 \rangle$, for all $\alpha \in N_2$, or $\# \to \# \ \langle 0 \rangle$, cannot be used to neutralize the negative energy $-(3i+1+k)$ associated with the rule $X \to Y$ due to the choice of $k$ as explained above. Hence, the only possibility is to correctly simulate the matrix $m_i$, by using simultaneously with $X \to Y \ \langle -(3i+1+k) \rangle$ the rule $A \to x \ \langle +(3i+1+k) \rangle$ with an energy balance of 0.

In the same way, in order to use a rule $X \to Y \ \langle -(3(s+j)+1+k) \rangle$ associated with a matrix $m_{s+j} : (X \to Y, A \to \#)$, $1 \leq j \leq t$, we have to use at the same step the corresponding rule $Z_j \to Z_j \ \langle +(3(s+j)+1+k) \rangle$, which means that $A$ is not present in the current multiset, or if $A$ is present, because of the maximality condition for the energy balance, the rule $A \to \# \ \langle +(3(s+j)+2+k) \rangle$ must be used, thus introducing the trap symbol $\#$, which can evolve forever by the rule $\# \to \# \ \langle 0 \rangle$. Therefore, the only continuation which does not lead to a computation which continues forever is that one which correctly simulates the use of the matrix $m_{s+j}$.

In conclusion, we have shown that $N_1(\Pi) = \Psi_T(L(G))$. □

**Theorem 2.** *$PsRE = ECP_{0,m}(1)$, for all $m \geq 1$.*

*Proof.* Adding the rules $a \to (a, out) \ \langle 0 \rangle$, for all $a \in T$, in the energy-controlled P system constructed in the preceding proof, already yields the desired result. □

In the previous proofs, the energy balance value 1 is only needed for simulating the matrices with appearance checking; hence, having no rules for simulating such matrices we immediately obtain:

**Corollary 1.** *$PsMAT \subseteq ECP_{1,m}(0) = ECP_{0,m}(0)$, for all $m \geq 1$.*

*Proof.* Having no rules for simulating matrices with appearance checking (and also omitting the rule $\# \to \# \ \langle 0 \rangle$) in the ECP system constructed in Theorem 1 (and in Theorem 2, respectively) yields an ECP system of the desired restricted form. Therefore, we need no additional symbols $Z_j$, hence, only

$f \to \lambda \ \langle -(3(s+t+1)+1+k) \rangle$ and

$Z \to \lambda \ \langle +(3(s+t+1)+1+k) \rangle$

are the rules we need at the end (in the last step) of a halting computation in $\Pi$. □

Whether this inclusion is strict or not remains as an unsolved problem. A candidate for showing the strictness would be the set $\{(2^n) \mid n \geq 1\}$, which is not the Parikh image of a matrix language (all one-letter matrix languages are regular, [6]).

Note that $PsCF \subset PsMAT$: while $PsCF$ is equal to the family of semilinear sets of vectors of natural numbers, $MAT$ contains non-semilinear languages. An example is $\{a^n b^m \mid n \geq 1, 1 \leq m \leq 2^n\}$ (see [2]).

ECP systems with 0 being the only allowed energy balance value in some sense correspond with stable systems, where any actions within the system have to conserve the total energy. Yet this is not only true for the ECP systems considered in Corollary 1, but also for the ECP systems constructed in the proofs of the preceding theorems: only those computations are successful which conserve the total energy in every step; having to apply a rule which would increase the total energy leads to a "crash" of the system, i.e., it will never halt.

Moreover, it is worth mentioning that the condition of maximal parallelism (there should not be another multiset $B$ of rules which is applicable to the objects in the underlying region with $B \supset A$ and yields the same energy balance) can be omitted for energy-controlled P systems without changing the results obtained so far in this paper.

We now show that ECP systems also allow us to generate any recursively enumerable language of strings as they appear as the sequences of terminal symbols sent out of the skin membrane of the system:

**Theorem 3.** *$RE = ECP_{2,m}(1)$, for all $m \geq 1$.*

*Proof.* We only give a sketch of the proof, because otherwise we had to go into deep details of proofs elaborated in [5]. In fact, there (graph-controlled as well as) matrix grammars for a string language $L \in RE$ work in that way to first generate a terminal word $w$ symbol by symbol from left to right while simultaneously generating a unary encoding of this word $w$, which finally is taken as input for (the simulation of) a register machine that halts if and only if $w \in L$. Therefore, when simulating such a matrix grammar by an ECP system $\Pi$ as described in the proof of Theorem 2, the rules $a \to (a, out)\ \langle 0 \rangle$, for all $a \in T$, send out the terminal symbols of the word $w$ in the correct ordered sequence, and the ECP system $\Pi$ halts if and only if $w \in L$. In conclusion, we have $N_2(\Pi) = L$. □

In this case, the condition of maximal parallelism is necessary to guarantee that the terminal symbols are sent out immediately as soon as possible so that they appear outside the skin membrane in the correct ordering.

The idea used in the proof of Theorem 3 also generalizes to arbitrary types $X$ of P systems that allow for the simulation of matrix grammars and guarantee that terminal symbols $a$ only are generated in the skin membrane in the correct sequence as they should form the string, and moreover, these terminal symbols are immediately sent out by rules of the form $a \to (a, out)$. Therefore, in many cases a result like that of Theorem 3 is obtained "for free" when already having proved a result like that of Theorem 2.

## 4 Energy-Controlled P Systems with Strings

Let us now consider the case when the objects of our systems are represented by strings and the evolution rules are rewriting rules. Formally, an *energy-controlled*

*P system* of this type is a construct

$$\Pi = (V, \mu, L_1, \ldots, L_m, R_1, \ldots, R_m, d_1, \ldots, d_m),$$

where:

1. $V$ is an alphabet;
2. $\mu$ is a membrane structure consisting of $m$ membranes, with the membranes and the regions labelled in a one-to-one manner with $1, 2, \ldots, m$; the skin membrane is labelled with 1;
3. $L_1, \ldots, L_m$ are multisets of strings over the alphabet $V$ associated with the regions $1, 2, \ldots, m$ of $\mu$;
4. $R_i, 1 \leq i \leq m$, are finite sets of *evolution rules* of the form $a \to v(tar)\ \langle e \rangle$, where $a \in V, v \in V^*$, *tar* is one of the target indications *here, out, in*, and $e$ is an integer number specifying the *energy* associated with the rule;
5. $d_1, \ldots, d_m$ are non-negative integers indicating the energy range of the corresponding membranes.

In each time unit and in each membrane of the system, each copy of a string can be rewritten by at most one rule. After rewriting a copy of the string $x$ by using a rule $a \to v(tar)\ \langle e \rangle$, that copy of $x$ is no longer present in the system, while the string resulting from rewriting is sent to the membrane indicated by *tar*, in the usual manner. Moreover, we have to take into account the energy restriction, in the same way as in the previous section: the total energy of rules used in membrane $i$, for rewriting the strings present in that membrane, has to be within the given range $[0, d_i]$, no extension of this multiset is possible such that more strings are affected without changing the energy balance, and the multiset is chosen in an optimal way such that no other multiset of rules can be applied to the present strings in membrane $i$ yielding a larger total energy within the energy range $[0, d_i]$ assigned to the membrane.

A computation is correctly finished only if it halts, i.e., if a configuration is obtained where no further evolution step is possible any more. The result of a halting computation either consists of all terminal strings found within the skin membrane in the final configuration or else of all terminal strings which are sent out of the system during the computation. In this way, an ECP system $\Pi$ generates a string language $L(\Pi)$ or $L'(\Pi)$, respectively, consisting of all strings which are produced as described above by all possible halting computations.

We denote by $ECP_m(d)$ and $ECP'_m(d)$ the family of string languages $L(\Pi)$ or $L'(\Pi)$, respectively, generated by ECP systems with at most $m$ membranes, $m \geq 1$, and with the maximal energy bound associated with a membrane at most $d$; if no bound on the number of membranes or on the maximal energy associated with each region is imposed, then we replace the corresponding parameters by $*$.

A result similar to Theorem 3 holds true for the case of string objects, too.

**Theorem 4.** $RE = ECP_m(1) = ECP'_m(1)$, *for all* $m \geq 1$.

*Proof.* The main idea is to take an ECP system $\Pi$ like that already constructed in the proof of Theorem 1 for a given matrix grammar $G$ and to replace the initial multiset $XAZZ_1 \dots Z_t$ by the three initial words $X$, $AZ$, and $Z_1 \dots Z_t$. For the case that the result of a halting computation are all the strings to be found in the skin membrane at the end of the computation, in that way we get an ECP system with $L(\Pi) = L(G)$. Hence, we conclude $RE = ECP_1(1)$.

For guaranteeing that a computation only halts if the word to be sent out does not contain any non-terminal symbol, some additional modifications are necessary.

Let $G = (N, T, (M, F), S)$ be a matrix grammar in the strong binary normal form, with $N = N_1 \cup N_2 \cup \{S, \#\}$. Assume that we have $s$ matrices of the form $(X \to Y, A \to x)$, with $X, Y \in N_1$, $x \in (N_2 \cup T)^*$, and $t$ matrices of the form $(X \to Y, A \to \#)$, $X, Y \in N_1$, $A \in N_2$. Moreover, let $(f \to \lambda)$ be the unique terminal matrix. We label the matrices $(X \to Y, A \to x)$ by $m_i$, $1 \leq i \leq s$, and the matrices of the form $(X \to Y, A \to \#)$ by $m_{s+j}$, $1 \leq j \leq t$.

We now construct the P system (of degree 1)

$$\Pi = (V, [_1]_1, L_1, R_1, 1),$$

with

$$V = N \cup T \cup \{Z\} \cup \{Z_j \mid 1 \leq j \leq t\} \cup \{X_j \mid 1 \leq j \leq t\},$$

$L_1 = \{X_0, A_0 Z, Z_1 \dots Z_t, \}$ for $(S \to X_0 A_0)$ being the initial matrix of $G$,

and the sets of rules $R_1$ containing the following rules:

1. For each matrix $m_i : (X \to Y, A \to x) \in M$, $1 \leq i \leq s$, with $X, Y \in N_1$, $x \in (N_2 \cup T)^*$, we introduce the rules
   $X \to Y \ \langle -(3i + 1 + k) \rangle$,
   $A \to x \ \langle +(3i + 1 + k) \rangle$.
2. For each matrix $m_{s+j} : (X \to Y, A \to \#) \in M$, $1 \leq j \leq t$, $X, Y \in N_1$, $A \in N_2$, we introduce the rules
   $X \to Y \ \langle -(3(s + j) + 1 + k) \rangle$,
   $A \to \# \ \langle +(3(s + j) + 2 + k) \rangle$,
   $Z_j \to Z_j \ \langle +(3(s + j) + 1 + k) \rangle$.
3. We also need the following rules to eventually keep the system from halting:
   $Z \to Z \ \langle 0 \rangle$,
   $\# \to \# \ \langle 1 \rangle$.
4. For "cleaning up" the additional symbols $Z$ and $Z_j$, $1 \leq j \leq t$, as well as to send out a terminal string, we finally use the following rules:
   $f \to X_1 \ \langle -(3(s + t + 1) + 1 + k) \rangle$,
   $Z \to (\lambda, out) \ \langle +(3(s + t + 1) + 1 + k) \rangle$
   $\alpha \to \# \ \langle +(3(s + t + 1) + 2 + k) \rangle$, for all $\alpha \in N_2$;
   $X_j \to X_{j+1} \ \langle -(3(s + t + j + 1) + 1 + k) \rangle$, $1 \leq j < t$,
   $Z_j \to \lambda \ \langle +(3(s + t + j + 1) + 1 + k) \rangle$, $1 \leq j < t$;
   $X_t \to \lambda \ \langle -(3(s + 2t + 1) + 1 + k) \rangle$,
   $Z_t \to \lambda \ \langle +(3(s + 2t + 1) + 1 + k) \rangle$.

For the constant $k$ we take $k := 6(s + 2t + 3)$; this again guarantees that when choosing a rule with a negative energy value only one of the rules with the corresponding positive value, eventually $+1$, can be chosen.

Assume that at some moment we have the strings $X$, $wZ$, $Z_1 \dots Z_t$; initially, $X = X_0$ and $w = A_0$, for $(S \to X_0 A_0)$ being the initial matrix of $G$.

If we use a rule $X \to Y \langle -(3i + 1 + k) \rangle$ associated with a matrix $m_i : (X \to Y, A \to x)$, for some $1 \leq i \leq s$, then at the same step we also have to use the rule $A \to x \langle +(3i + 1 + k) \rangle$. Indeed, none of the other rules associated with positive energy values $> 1$ nor any of the rules $Z \to Z \langle 0 \rangle$ or $\# \to \# \langle 1 \rangle$ can be used to neutralize the negative energy $-(3i+1+k)$ associated with the rule $X \to Y$ due to the choice of $k$ as explained above. Hence, the only possibility is to correctly simulate the matrix $m_i$, by using simultaneously with $X \to Y \langle -(3i+1+k) \rangle$ the rule $A \to x \langle +(3i + 1 + k) \rangle$ with an energy balance of 0. To the string $Z_1 \dots Z_t$ no rule can be applied in this case.

In a similar way, in order to use a rule $X \to Y \langle -(3(s+j)+1+k) \rangle$ associated with a matrix $m_{s+j} : (X \to Y, A \to \#)$, $1 \leq j \leq t$, we have to use at the same step the corresponding rule $Z_j \to Z_j \langle +(3(s + j) + 1 + k) \rangle$ affecting the string $Z_1 \dots Z_t$, which means that $A$ is not present in the word $w$ (in that case, $Z \to Z \langle 0 \rangle$ is applied to the string $wZ$) or if $A$ is present in $w$, because of the maximality condition for the energy balance, the rule $A \to \# \langle +(3(s+j)+2+k) \rangle$ must be used, thus introducing the trap symbol $\#$, which can evolve forever by the rule $\# \to \# \langle 1 \rangle$, whereas in this case no rule is applied to the string $Z_1 \dots Z_t$. Therefore, the only continuation which does not lead to a computation which continues forever is that one which correctly simulates the use of the matrix $m_{s+j}$.

As long as the additional symbol $Z$ will exist, at least the rule $Z \to Z \langle 0 \rangle$ can be used, hence the computation cannot stop; yet $Z$ can only be removed in combination with $f$; remember that $f$ - on the left-hand side of a production in $G$ - only appears in the terminal matrix $(f \to \lambda)$. Before sending out a word it has to be checked that no non-terminal symbols occur in this word, otherwise the computation should not stop, which is guaranteed by the rules $\alpha \to \# \langle +(3(s+t+1)+2+k) \rangle$, for all $\alpha \in N_2$; an application of such a rule will block the elimination of the symbol $Z$ and then cause the system not to halt because of the introduction of the trap symbol $\#$. Therefore, a successful computation has to lead to a configuration with $f$, $wZ$, $Z_1 \dots Z_t$ being the contents of the skin membrane for some $w \in T^*$. By using the rules

$f \to X_1 \langle -(3(s + t + 1) + 1 + k) \rangle$,
$Z \to (\lambda, out) \langle +(3(s + t + 1) + 1 + k) \rangle$;
$X_j \to X_{j+1} \langle -(3(s + t + j + 1) + 1 + k) \rangle$, $1 \leq j < t$,
$Z_j \to \lambda \langle +(3(s + t + j + 1) + 1 + k) \rangle$, $1 \leq j < t$;
$X_t \to \lambda \langle -(3(s + 2t + 1) + 1 + k) \rangle$,
$Z_t \to \lambda \langle +(3(s + 2t + 1) + 1 + k) \rangle$

in this sequence (here the energy balance is always 0) we finally have no non-empty string in the skin membrane any more after having sent out the terminal word $w$, and the computation halts.

In conclusion, we have shown that $L'(\Pi) = L(G)$, and therefore $RE = ECP'_1(1)$, too. □

As the construction of the ECP system above shows, we could only consider sets of rules and sets of strings instead of multisets and the proof would still work, because we have the condition that at most one rule can be applied to each word in one computation step.

The following corollary follows from Theorem 4 as Corollary 1 followed from Theorem 1:

**Corollary 2.** *$MAT \subseteq ECP_m(0)$, for all $m \geq 1$.*

In contrast to Corollary 1, it remains an open question whether $MAT \subseteq ECP'_m(0)$, *for all* $m \geq 1$, too, because in the proof of the preceding theorem we had to check whether the word sent out still contains a non-terminal symbol or not by using rules like those for simulating appearance checking.

## 5 Conclusion

The interested reader may compare the proofs given in [13] for P systems with energy accounting with the proofs given in this paper for energy-controlled P systems. In some sense, not violating the balanced energy level 0 for the involved energies in energy-controlled P systems corresponds with the fact that there successful computations have to avoid the premature dissolvation of the second membrane and, moreover, even allows to avoid the use of catalysts, whereas the priorities needed in P systems with energy accounting (needed for simulating the appearance checking) have their counterpart in the maximality condition for the sum of energies in energy-controlled P systems (i.e., more total energy means a higher priority).

Hence, the (quite natural and biochemically well motivated) feature used for controlling the energy in energy-controlled P systems turns out to be very powerful from a computational point of view: characterizations of recursively enumerable languages in the case of string objects and characterizations of recursively enumerable sets of vectors of natural numbers in the case of symbol objects are obtained with only one membrane and only two energy values 0 and 1.

**Acknowledgements**

I am very grateful to Gheorghe Păun for all his inspiring ideas.

## References


1. Calude, C.S., Păun, Gh.: Computing with Cells and Atoms (Chapter 3: "Computing with Membranes"). Taylor & Francis, London (2001)
2. Dassow, J., Păun, Gh.: Regulated Rewriting in Formal Language Theory. Springer-Verlag, Berlin (1989)
3. Dassow, J., Păun, Gh.: On the Power of Membrane Computing. Journal of Universal Computer Science **5**, 2 (1999) 33–49 (`http://www.iicm.edu/jucs`)
4. Freund, R., Freund, F.: Molecular computing with generalized homogeneous P systems. In: Condon, A., Rozenberg, G. (eds.): Proc. Conf. DNA6, Leiden (2000) 113–125
5. Freund, R., Păun, Gh.: On the Number of Non-terminals in Graph-controlled, Programmed, and Matrix Grammars. In: Margenstern, M., Rogozhin, Y. (eds.): Proc. Conf. Universal Machines and Computations, Chişinău (2001). Springer-Verlag, Berlin (2001)
6. Hauschild, D., Jantzen, M.: Petri nets algorithms in the theory of matrix grammars. Acta Informatica **31** (1994) 719–728
7. The P Systems Web Page: `http://psystems.disco.unimib.it/`
8. Păun, Gh.: Computing with Membranes. Journal of Computer and System Sciences **61,** 1 (2000) 108–143 and TUCS Research Report 208 (1998) (`http://www.tucs.fi`)
9. Păun, Gh.: Computing with Membranes: An Introduction. Bulletin EATCS **67** (1999) 139–152
10. Păun, Gh.: Membrane Computing: An Introduction. Springer-Verlag, Berlin (2002)
11. Păun, Gh.: Computing with membranes (P systems): Twenty six research topics. Auckland University, CDMTCS Report 119 (2000) (`http://www.cs.auckland.ac.nz/CDMTCS`)
12. Păun, Gh., Rozenberg, G., Salomaa, A.: Membrane computing with external output. Fundamenta Informaticae **41**, 3 (2000) 259–266, and TUCS Research Report 218 (1998) (`http://www.tucs.fi`)
13. Păun, Gh., Suzuki, Y., Tanaka, H.: P Systems with energy accounting. Int. J. Computer Math. **78**, 3 (2001) 343–364
14. Salomaa, A., Rozenberg, G. (eds.): Handbook of Formal Languages. Springer-Verlag, Berlin (1997)

# P Systems with Activated/Prohibited Membrane Channels

Rudolf Freund and Marion Oswald

Department of Computer Science
Technische Universität Wien, Wien, Austria
{rudi,marion}@emcc.at

**Abstract.** We investigate a variant of purely communicating P systems, where multisets of activators can open channels for certain objects to pass through membranes in one direction; however, the permeability of a channel can be controlled by multisets of prohibitors, too.
We will show that for such systems with only one membrane and using only singleton activator and prohibitor sets, we already obtain universal computational power. When using systems with activating multisets for membrane channels only, we obtain a similar result. By showing a close correspondence to P systems with symport/antiport as introduced in [13] we can optimize some results given there.

## 1 Introduction

P systems were introduced in [10] by Gh. Păun as distributed parallel computing devices that are abstracted from cell functioning. The most important features considered in the various models of P systems investigated so far (e.g., see [3], [4], [10], [11], for a comprehensive overview see [12]) are the membrane structure and specific features of the membranes, especially for the transfer of objects through the membranes. A *membrane structure* consists of membranes hierarchically embedded in the outermost *skin membrane*; every membrane encloses a *region* possibly containing other membranes; the part delimited by the membrane labelled by $k$ and its inner membranes is called *compartment* $k$. All membranes can be labelled by natural numbers (starting with 1 for the skin membrane) so that a membrane structure can uniquely be described by a string of correctly matching parentheses with each pair corresponding to a membrane. In the membranes, multisets of objects can be placed, which evolve according to given evolution rules. Applying the latter ones in a non-deterministic, maximally parallel way, the system passes from one configuration to another one, thereby performing a computation. Only halting computations produce a result, which consists of the objects present in a specified output membrane.

In contrast to various other models of P systems, where the objects themselves can be transformed during a computation, we consider purely communicating systems, as already done, e.g., in [13].

In the model presented here, objects can cross the membranes by passing through corresponding channels that have been opened by means of activators,

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 261–269, 2003.

unless the channel is blocked by a prohibitor. Thus we have activating and prohibiting rules that are used in a non-deterministic maximally parallel manner. Applying an activating rule means that an activator multiset (or a single activator symbol) opens input and output channels for specific objects. In the following substep, each object can pass through the surrounding membrane provided there is no prohibitor active, which prevents the object from passing through the corresponding channel. In some variants of P systems with activated/prohibited membrane channels, not for all the channels opened, an object for passing needs to be present; but anyway, whether they have been used or not, the channels will be closed again in the subsequent step.

In contrast to [13], we assume the environment to contain all objects in arbitrarily many copies.

In the following section we first give some preliminary definitions and define $n$-register machines, the universal model of computation we use for proving our new results elaborated in this paper; in the third section we introduce P systems with activated/prohibited membrane channels, followed by an example elaborated in the succeeding section. In the fifth section we show that different restricted variants of these systems with only one membrane can simulate $n$-register machines quite easily, which proves their universal computational power. Finally we can optimize some results established in [13] for P systems with symport/antiport by interpreting these P systems as a special variant of the P systems with activated/prohibited membrane channels as introduced in this paper. Independently, the same improvements were achieved in [8].

## 2 Preliminary Definitions

The set of non-negative integers is denoted by $\mathbf{N}_0$, the set of positive integers by $\mathbf{N}$. An *alphabet* $V$ is a finite non-empty set of abstract *symbols*. Given $V$, the free monoid generated by $V$ under the operation of concatenation is denoted by $V^*$; the *empty string* is denoted by $\lambda$, and $V^* \setminus \{\lambda\}$ is denoted by $V^+$. A multiset over $V$ is represented as string over $V$ (and any of its permutations). By $\mid x \mid$ we denote the length of the word $x$ over $V$ as well as the number of elements in the multiset represented by $x$. For more notions from the theory of formal languages, the reader is referred to [2].

As we mostly will restrict ourselves to consider multisets of symbols, our main purpose will be to consider computational models for Parikh sets of recursively enumerable languages, especially for representations of recursively enumerable sets of non-negative integers. A well-known example for such a - very simple - mechanism are register machines (see [9] for some original definitions and [5], [15] for definitions like that we use in this paper).

An *$n$-register machine* is a construct

$$RM = (n, R, i, h)$$

where

- $n$ is the number of registers,
- $R$ is a set of labelled instructions of the form $j : (op(r), k, l)$, where $op(r)$ is an operation on register $r$ of $RM$, $j, k, l$ are labels from the set $Lab(RM)$ (which numbers the instructions in a one-to-one manner),
- $i$ is the initial label, and
- $h$ is the final label.

The machine is capable of the following instructions:

**(A(r),k,l)** Add one to the contents of register $r$ and proceed to instruction $k$ or to instruction $l$; in the deterministic variants usually considered in the literature we demand $k = l$.

**(S(r),k,l)** If register $r$ is not empty then subtract one from its contents and go to instruction $k$, otherwise proceed to instruction $l$.

**HALT** Stop the machine. This instruction can only be assigned to the final label $h$.

In their *deterministic variant*, such $n$-register machines can be used to compute any partial recursive function $f : \mathbf{N}_0 \to \mathbf{N}_0$; starting with $n \in \mathbf{N}_0$ in register 1, $RM$ has computed $f(n) = r$ if it halts in the final label $h$ with register 1 containing $r$. If the final label cannot be reached, $f(n)$ remains undefined.

A deterministic $n$-register machine can also analyze an input $n \in \mathbf{N}_0$ in register 1, which is accepted if the register machine finally stops by the halt instruction with all its registers being empty. If the machine does not halt, the analysis was not successful.

In their *non-deterministic variant*, $n$-register machines can compute any recursively enumerable set of non-negative integers. Starting with all registers being empty, we consider a computation of the $n$-register machine to be successful, if it halts (with the result being contained in the first register and with all other register being empty).

The results proved in [5] and [7] immediately lead us to the following result which the proofs elaborated in this paper are based on:

**Proposition 1.** *For any recursively enumerable set of non-negative integers $L$ there exists a non-deterministic* 3*-register machine $M$ generating $L$.*

## 3 P Systems with Activated/Prohibited Membrane Channels

A *P system with activated/prohibited membrane channels* is a construct $\Pi$ of the following form:

$$\Pi = (V, \mu, w_1, ..., w_n, R_1, ..., R_n, i_0)$$

where

- $V$ is an alphabet of *objects*;

- $\mu$ is a *membrane structure* (with the membranes labelled by natural numbers $1, ..., n$ in a one-to-one manner);
- $w_1, ..., w_n$ are multisets over $V$ associated with the compartments $1, ..., n$ of $\mu$;
- $R_1, ..., R_n$ are finite sets of *rules* associated with the compartments $1, ..., n$, which can be of the following forms:
  1. *activating rules*: $\langle P; x, out; y, in \rangle$, where $x, y \in V^*$ and $P$ is a finite multiset over $V$,
  2. *prohibiting rules*: $\langle b, out; Q \rangle$ or $\langle b, in; Q \rangle$, where $b \in V$ and $Q$ is a finite multiset over $V$.
- $i_0 \in \{1, ..., n\}$ is the label of the *output membrane*.

A system that uses only activating rules is called a *P system with activated membrane channels*.

Starting from the *initial configuration*, which consists of $\mu$ and $w_1, ..., w_n$, the system passes from one configuration to another one by non-deterministically in a maximally parallel way choosing rules from $R_i$ and applying them in the following sense:

Let $x = x_1...x_m$ and $y = y_1...y_n$. An activating rule $\langle P; x, out; y, in \rangle$ means that by the activator multiset $P$ an output channel for each symbol $x_i$, $1 \leq i \leq m$, is activated, and for each $y_j$, $1 \leq j \leq n$, an input channel is activated. In the following substep of a derivation (computation), each activated channel allows for the transport of one object $x_i$ and $y_j$, respectively, provided there is no prohibitor multiset $Q$ active by a prohibiting rule $\langle x_i, out; Q \rangle$ or $\langle y_j, in; Q \rangle$, respectively (which means that the multiset $Q$ can be found in the underlying compartment). The activating multisets $P$ in the activating rules have to be chosen in a maximally parallel way.

A sequence of transitions is called a *computation*; it is *successful*, if and only if it halts. The *result* of a successful computation then is considered to be the multiset of objects present in the designated output membrane $i_0$. A non-halting computation does not produce a result.

## 4 Example

We consider the following example. Let $\Pi = (V, [_1]_1, 1a, R, 1)$, where:

$V = \{1, 1', 1'', 2, 2', 2'', a, b, f_1, f_1', f_2, f_2'\}$ and

$$
\begin{aligned}
R_1 = \{ & \langle 1; 1a, out; 1bbf_1, in \rangle, \ \langle f_1, in; a \rangle, \ \langle f_1; f_1, out; f_1, in \rangle, \\
& \langle 1; 1, out; 1'1'', in \rangle, \ \langle 1'', in; a \rangle, \\
& \langle 1'; 1'1'', out; 2f_1', in \rangle, \ \langle f_1', in; 1'' \rangle, \ \langle f_1'; f_1', out; f_1', in \rangle, \\
& \langle 1'; 1'1'', out; f_1', in \rangle, \\
& \langle 2; 2b, out; 2af_2, in \rangle, \ \langle f_2, in; b \rangle, \ \langle f_2; f_2, out; f_2, in \rangle, \\
& \langle 2; 2, out; 2'2'', in \rangle, \ \langle 2'', in; b \rangle, \\
& \langle 2'; 2'2'', out; 1f_2', in \rangle, \ \langle f_2', in; 2'' \rangle, \ \langle f_2'; f_2', out; f_2', in \rangle \}
\end{aligned}
$$

Starting with the multiset $1a$ in the skin membrane, the activator 1 can get active. By using $\langle 1; 1a, out; 1bbf_1, in\rangle$, it can open output channels for 1 and $a$, as well as input channels for 1 and $bb$. (As the prohibitor $a$ is present, the input channel for $f_1$ will not be opened because of the prohibiting rule $\langle f_1, in; a\rangle$.) In the following substep, $a$ and 1 leave the system, whereas the multiset $1bb$ can enter the skin membrane. Going on with $\langle 1; 1, out; 1'1'', in\rangle$ and $\langle 1'; 1'1'', out; 2f_1', in\rangle$ we obtain the configuration $2bb$.

Now 2 can open channels for 2 and $b$ to pass out and for 2 and $a$ to pass in. Hence, for every $b$ leaving the system, one $a$ can enter. Having sent out all copies of $b$, we can proceed with $\langle 2; 2, out; 2'2'', in\rangle$ and $\langle 2'; 2'2'', out; 1f_2', in\rangle$. In a similar way as before, the input channel for $f_2'$ is prevented from being opened, so that the system now contains $1aa$. We can proceed successfully by sending out all copies of $a$ while letting twice as many copies of $b$ in; every $b$ can move out again, allowing for an $a$ to enter, and so on.

If in any moment, after just having sent out all occurrences of $a$, the activating rule $\langle 1'; 1'1'', out; f_1', in\rangle$ instead of $\langle 1'; 1'1'', out; 2f_1', in\rangle$ is used, the system reaches a halting configuration. The result can then be found as $b^{2^n}$ in the skin membrane. In any other case, the system would, sooner or later, get stuck in an endless loop with either $\langle f_i; f_i, out; f_i, in\rangle$ or $\langle f_i'; f_i', out; f_i', in\rangle$, $i \in \{1, 2\}$, and consequently not produce a useful result. In conclusion, we obtain $L(\Pi) = \left\{b^{2^n} \mid n \geq 1\right\}$.

As can be seen from the construction of the rules in $\Pi$, the activating rules are of the forms $\langle l; l, out; y, in\rangle$ or $\langle l; lx, out; yf, in\rangle$ with $x, l, f \in V$ and $y \in V^*$, where for $f$ we have the rule $\langle f; f, out; f, in\rangle$, which leads to a non-halting computation if $f$ is allowed to enter through the skin membrane, as well as the prohibiting rule $\langle f, in; x\rangle$. This guarantees that the channel for $x$ opened by the rule $\langle l; lx, out; yf, in\rangle$ has to be used for sending out an object $x$, i.e., an object $x$ must be present in the skin membrane, otherwise the failure symbol (trap symbol) $f$ can enter (leading to a non-halting computation). Observe that due to our definition non-prohibited activated input channels are always used, because in the environment, all symbols are available in an unlimited number.

## 5 Results

Based on the proof techniques used, e.g., in [5], [6], we can immediately show the following results using Proposition 1.

**Theorem 1.** *Let $L \subseteq \mathbf{N}_0$ be a recursively enumerable set of non-negative integers. Then $L$ can be generated by a P system with activated/prohibited membrane channels in only one membrane using only singleton activators and prohibitors; moreover, a non-prohibited activated output channel has to be used only if the corresponding symbol is present in the skin membrane.*

*Proof. (Sketch.)* According to Proposition 1, we only have to elaborate how we can simulate the instructions of a 3-register machine:

- An Add-instruction $j : (A(i), k, l)$ is simulated by the two activating rules $\langle j; j, out; ka_i, in\rangle$ and $\langle j; j, out; la_i, in\rangle$.
- A conditional Subtract-instruction $j : (S(i), k, l)$ is simulated by the following rules:

$\langle j; ja_i, out; kf_j, in\rangle$ $\langle f_j, in; a_i\rangle$
$\langle j; j, out; j'j'', in\rangle$ $\langle j'', in; a_i\rangle$
$\langle j'; j'j'', out; lf_j, in\rangle$ $\langle f_j, in; j''\rangle$
$\langle f_j; f_j, out; f_j, in\rangle$

- The halting instruction $h : HALT$ is simulated by the rule $\langle h; h, out; \lambda, in\rangle$.

As already argued at the end of the preceding section in the example given there, the construction of the rules in the P system with activated/prohibited membrane channels guarantees that rules sending out another object together with the activating symbol can only be used without introducing a failure symbol (trap symbol) if also this other object is present in the skin membrane. This means that all activated output channels will be used in halting computations. This observation completes the proof. □

Looking carefully into the functioning of how the P system with activated / prohibited membrane channels constructed in the preceding proof simulates the instructions of the underlying $n$-register machine, we reveal the surprising and unexpected fact that we need not demand the activating rules to be chosen in a maximally parallel manner; it is sufficient to apply the rules in a sequential way.

In conclusion, the observations considered above show that P systems with activated/prohibited membrane channels could also be interpreted as generalized P systems [4] using the rules in a sequential way only. In that case, the labels $j$ from the simulated register machine, which are used as activating symbols, correspond to the ground symbols used in generalized P systems (ground symbols are "consumed" when being used in an evolution rule, the activating labels are "consumed" by being sent out into the environment).

On the other hand, we can even avoid the use of prohibitor rules, if we make use of activator multisets instead of single activator symbols, which, in rules of the form $\langle x; x, out; y, in\rangle$, is equivalent to demand that every activated output channel has to be used in the subsequent substep. For sake of conciseness, such activating rules of the form $\langle x; x, out; y, in\rangle$ in the following will be written in a shorter way as $(x, out; y, in)$.

**Theorem 2.** *Let $L \subseteq \mathbf{N}_0$ be a recursively enumerable set of non-negative integers. Then $L$ can be generated by a P system with activated membrane channels that consists of the simplest membrane structure.*

*Proof. (Sketch.)* Again, we only have to show how the instructions of an $n$-register machine (due to Proposition 1, $n = 3$ is sufficient) can be simulated:

- An Add-instruction $j : (A(i), k, l)$ is simulated by the two activating rules $(j, out; ka_i, in)$ and $(j, out; la_i, in)$.
- A conditional Subtract-instruction $j : (S(i), k, l)$ is simulated by the following rules:

  $(ja_i, out; k, in)$
  $(j, out; j'j'', in)$
  $(j'a_i, out; f, in)$ $(f, out; f, in)$
  $(j'', out; j''', in)$
  $(j'j''', out; l, in)$

- The halting instruction $h : HALT$ is simulated by the rule $(h, out; \lambda, in)$.

In this case, the condition of maximal parallelism guarantees that the rule $(j'a_i, out; f, in)$ is applied in parallel with $(j'', out; j''', in)$, which leads to a non-halting computation by the introduction of the failure symbol (trap symbol) $f$. Only if in the current configuration no symbol $a_i$ is present in the skin membrane, the object $j'$ can wait one step for being used in the rule $(j'j''', out; l, in)$ together with the symbol $j'''$ introduced by the rule $(j'', out; j''', in)$. □

If we compare the notions introduced for P systems with activated membrane channels as defined above with the definitions of P systems with symport/antiport as introduced in [13], we realize that P systems with activated membrane channels and rules of the form $(x, out; y, in)$, $x, y \in V^*$, can be interpreted as P systems with symport/antiport (rules). The *radius* of an antiport rule $(x, out; y, in)$, where $x, y \in V^+$, is the pair of numbers $(|x|, |y|)$; the radius of a symport rule of the form $(z, out; \lambda, in)$ or $(\lambda, out; z, in)$ with $z \in V^+$ is $|z|$.

Hence, small modifications of the construction given in the preceding proof allow us to considerably improve Theorem 4.1 in [13] by reducing the number of membranes to 1 and the radius of the antiport rules to $(2, 1)$, or $(1, 2)$, respectively.

In contrast to P systems with antiport rules as defined in [13], we need not specify the environment, because we assume every symbol to appear in an unlimited number there.

**Theorem 3.** *Let $L \subseteq \mathbf{N}$ be a recursively enumerable set of positive integers. Then $L$ can be generated by a P system with activated membrane channels that consists of the simplest membrane structure and only uses rules of the form $(x, out; y, in)$ with the radius of the rules being $(2, 1)$ or $(1, 2)$, respectively.*

*Proof. (Sketch.)* In the proof constructed in the preceding theorem, we only have to replace the rule $(f, out; f, in)$ by the two rules $(f, out; f'f'', in)$ and $(f'f'', out; f, in)$ as well as the rule $(h, out; \lambda, in)$ by the rules $(h, out; h'h'', in)$ and $(h'h'', out; a_1, in)$. Now every rule obeys to the conditions for the radius to be $(2, 1)$ or $(1, 2)$, respectively. The rule $(h'h'', out; a_1, in)$ is the reason why we can only generate sets of positive integers. □

The following result for P systems with antiport rules immediately follows from the preceding theorem:

**Corollary 1.** *Let $L \subseteq \mathbf{N}$ be a recursively enumerable set of positive integers. Then $L$ can be generated by a P system with antiport rules that consists of the simplest membrane structure and only uses rules of the form $(x, out; y, in)$ with the radius of the rules being $(2,1)$ or $(1,2)$, respectively.*

The result stated above optimizes Theorem 4.1 in [13]; the P systems with antiport rules considered in Corollary 1 are not only optimal with respect to the number of membranes but also with respect to the radius of the rules used in the system: Using only rules with radius $(1,1)$ and $(2,1)$ in the simplest membrane structure we only get finite sets of non-negative integers. If we only allow rules with radius $(1,1)$ and $(1,2)$ we at most get sets of non-negative integers representing monotonic one-letter languages.

One additional symport rule of radius 1 is needed for generating any recursively enumerable set of non-negative integers:

**Corollary 2.** *Let $L \subseteq \mathbf{N}_0$ be a recursively enumerable set of non-negative integers. Then $L$ can be generated by a P system with symport/antiport that consists of the simplest membrane structure and only uses antiport rules of the form $(x, out; y, in)$ with the radius of the rules being $(2,1)$ or $(1,2)$, respectively, and only one single symport rule of radius 1.*

*Proof.* In the proof we constructed in the preceding theorem and corollary, respectively, we only have to replace the antiport rules $(h, out; h'h'', in)$ and $(h'h'', out; a_1, in)$ again by the symport rule $(h, out; \lambda, in)$. □

We want to point out that, independently, the same improvements for P systems with symport/antiport as established in Corollaries 1 and 2 were achieved in [8].

## 6 Conclusion

We have investigated P systems with activated/prohibited membrane channels which surprisingly already obtain universal computational power with the simplest membrane structure and only singleton activators and inhibitors. Even more unexpected, the rules need not be applied in the maximally parallel manner, but only in a sequential way.

If we use multisets for activating channels, we need no prohibiting multisets, but now the maximal parallelism for choosing the rules to be applied is essential.

Moreover, we should like to mention that rather similar ideas like that used with respect to activating/prohibiting membrane channels can be found in [1].

Finally, establishing the correspondence between a special variant of P systems with activated membrane channels and P systems with antiport rules, we have achieved an optimal result for P systems with antiport rules, which already obtain universal computational power with only one membrane and with rules

of radius $(2,1)$ or $(1,2)$, respectively, for recursively enumerable sets of positive integers. For generating recursively enumerable sets of non-negative integers, we also need a symport rule of radius 1. Independently, exactly the same results were also obtained by Frisco and Hoogeboom, see [8].

## References


1. Bernardini, F., Manca, V.: P Systems with Boundary Rules. In: [14] 97–102
2. Dassow, J., Păun, Gh.: Regulated Rewriting in Formal Language Theory. Springer-Verlag, Berlin (1989)
3. Dassow, J., Păun, Gh.: On the power of membrane computing. Journal of Universal Computer Science **5**, 2 (1999) 33–49 (`http://www.iicm.edu/jucs`)
4. Freund, R.: Sequential P systems. In: Freund, R. (ed.): Theorietag 2000, TU Wien (2000) 177–183
5. Freund, R., Oswald, M.: GP Systems with Forbidding Context. Fundamenta Informaticae **49**, 1-3 (2002) 81–102
6. Freund, R., Păun, Gh.: On the Number of Non-terminals in Graph-controlled, Programmed, and Matrix Grammars. In: Margenstern, M., Rogozhin, Y. (eds.): Proc. Conf. Universal Machines and Computations, Chişinău (2001). Springer-Verlag, Berlin (2001)
7. Freund, R., Păun, Gh.: From Regulated Rewriting to Computing with Membranes: Collapsing Hierarchies; *to appear in TCS*
8. Frisco, P., Hoogeboom, H.J.: Simulating counter automata by P systems with symport/antiport. In: [14] 237–248
9. Minsky, M. L.: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, New Jersey (1967)
10. Păun, Gh.: Computing with Membranes. Journal of Computer and System Sciences **61,** 1 (2000) 108–143 and TUCS Research Report 208 (1998) (`http://www.tucs.fi`)
11. Păun, Gh.: Computing with Membranes: An Introduction. Bulletin EATCS **67** (1999) 139–152
12. Păun, Gh.: Membrane Computing: An Introduction. Springer-Verlag, Berlin (2002)
13. Păun, A., Păun, Gh.: The Power of Communication: P Systems with Symport/Antiport. New Generation Computing **20,** 3 (2002) 295–306
14. Păun, Gh., Zandron, C. (eds.): Pre-Proceedings of Workshop on Membrane Computing (WMC-CdeA2002), Curtea de Argeş, Romania (2002)
15. Sosík, P.: P systems versus register machines: two universality proofs. In: [14] 371–382

# Membrane Systems with Symport/Antiport Rules: Universality Results

Rudolf Freund[1] and Andrei Păun[2]

[1] Department of Computer Science, Technische Universität Wien
Karlsplatz 13, A-1040 Wien, Austria
rudi@logic.at
[2] Department of Computer Science, University of Western Ontario
London, Ontario, N6A 5B7, Canada
apaun@csd.uwo.ca

**Abstract.** Symport and antiport are biological ways for transporting molecules through membranes in a "collaborating" manner; in the case of symport several molecules pass in the same direction, in the case of antiport two or more molecules pass in opposite directions. In this paper we first survey the results on the computing power of membrane systems (P systems) using only symport/antiport rules and then improve some of the results known so far. A recent variant of P systems with purely communicating rules introduced in [24] with the name of communicating P systems is revisited and optimal (with respect to the number of membranes) universality results for that particular variant are obtained, too.

## 1 Introduction

Recently in the area of membrane systems (usually called P systems) purely communicating P systems where the objects only pass through membranes, but are not affected by rules during a computation of the system, have become of great interest. A comprehensive overview on membrane systems (P systems) is given in the monograph [19]. An actual catalogue of publications and open questions is available on the web [13].

P systems with symport/antiport first were introduced in [14]; new results on this kind of membrane systems can be found in [9], [10], [15], [16], and [20]. Independently, several new results comparable with those eloborated in this paper for P systems with symport/antiport were obtained in [8], where the proofs are based on results for the model of counter automata (just another name for the register machines we use in this paper); instead, the proof for the corresponding result on P systems with symport rules given in this paper is based on the Z-binary normal form for matrix grammars.

In [23] and [24] another variant of purely *communicating P systems* was introduced; whereas in the universality proof given in [23], no bound on the number of membranes was given, we now show that the simplest membrane structure is sufficient. Moreover, we show how communicating P systems with only one

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 270–287, 2003.

membrane can be interpreted as P systems with symport/antiport, which allows us to establish optimal universality results for P systems with symport/antiport as a consequence of the corresponding optimal (with respect to the number of membranes) universality result for communicating P systems; independently, these optimal universality results for P systems with symport/antiport have been shown in [5] and [8].

## 2 Preliminaries

Before proceeding to a formal description of register machines and matrix grammars, we fix some basic notations first. For an alphabet $V$, by $V^*$ we denote the free monoid generated by $V$ under the operation of concatenation; the *empty string* is denoted by $\lambda$, and $V^* \setminus \{\lambda\}$ is denoted by $V^+$. Any subset of $V^+$ is called a *$\lambda$-free (string) language*. Moreover, by $\mathbf{N}_0$ we denote the set of non-negative integers and by $N_0^\beta RE$ ($N_1^\beta RE$) we denote the family of recursively enumerable sets of $\beta$-vectors $(y_1, ..., y_\beta)$ of non-negative (positive) integers. The family of recursively enumerable languages is denoted by $RE$.

In the following we will not distinguish between a vector $(y_1, ..., y_\beta)$, its representation by a multiset or its representation by a string with Parikh vector $(y_1, ..., y_\beta)$.

For more notions as well as basic results from the theory of formal languages, the reader is referred to [3], [11], and [22].

### 2.1 Register Machines

In this subsection we briefly recall the concept of Minsky's register machine. Minsky showed (e.g., see [12]) that the universal computational power can be reached by such an abstract machine using a finite number of registers for storing arbitrarily large non-negative integers. The machine runs a program consisting of numbered instructions of several simple types. Several variants of the machine with different number of registers and different instruction sets were shown to be computationally universal (e.g., see [12] for some original definitions and [4] for the definitions we use in this paper).

An *$n$-register machine* is a construct $M = (n, P, i, h)$, where:

- $n$ is the number of registers,
- $P$ is a set of labelled instructions of the form $j : (op(r), k, l)$, where $op(r)$ is an operation on register $r$ of $M$, $j, k, l$ are labels from the set $Lab(M)$ (which numbers the instructions in a one-to-one manner),
- $i$ is the initial label, and
- $h$ is the final label.

The machine is capable of the following instructions:

**(A(r),k,l)** Add one to the contents of register $r$ and proceed to instruction $k$ or to instruction $l$; in the deterministic variants usually considered in the literature we demand $k = l$.

**(S(r),k,l)** If register $r$ is not empty, then subtract one from its contents and go to instruction $k$, otherwise proceed to instruction $l$.

**HALT** Stop the machine. This additional instruction can only be assigned to the final label $h$.

In their *deterministic variant*, such $n$-register machines can be used to compute any partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$; starting with $(n_1, ..., n_\alpha) \in \mathbf{N}_0^\alpha$ in registers 1 to $\alpha$, $M$ has computed $f(n_1, ..., n_\alpha) = (r_1, ..., r_\beta)$ if it halts in the final label $h$ with registers 1 to $\beta$ containing $r_1$ to $r_\beta$. If the final label cannot be reached, then $f(n_1, ..., n_\alpha)$ remains undefined.

A deterministic $n$-register machine can also analyse an input $(n_1, ..., n_\alpha) \in \mathbf{N}_0^\alpha$ in registers 1 to $\alpha$, which is recognized if the register machine finally stops by the halt instruction with all its registers being empty. If the machine does not halt, then the analysis was not successful.

In their *non-deterministic variant*, $n$-register machines can compute any recursively enumerable set of non-negative integers (or of vectors of non-negative integers). Starting with all registers being empty, we consider a computation of the $n$-register machine to be successful, if it halts with the result being contained in the first ($\beta$) register(s) and with all other registers being empty.

The results proved in [4] (based on the results established in [12]) as well as in [6] and [7] immediately lead us to the following results:

**Proposition 1.** *For any partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$ there exists a deterministic $(\max\{\alpha, \beta\} + 2)$-register machine $M$ computing $f$ in such a way that, when starting with $(n_1, ..., n_\alpha) \in \mathbf{N}_0^\alpha$ in registers 1 to $\alpha$, $M$ has computed $f(n_1, ..., n_\alpha) = (r_1, ..., r_\beta)$ if it halts in the final label $h$ with registers 1 to $\beta$ containing $r_1$ to $r_\beta$ (and with all other registers being empty); if the final label cannot be reached, then $f(n_1, ..., n_\alpha)$ remains undefined.*

**Proposition 2.** *For any recursively enumerable set $L \subseteq \mathbf{N}_0^\beta$ 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 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 $(n_1, ..., n_\beta) \in L$.*

### 2.2 Matrix Grammars

In this subsection we recall the Z-binary normal form for matrix grammars, which will be used in some of the following universality proofs.

Consider a matrix grammar with appearance checking $G = (N, T, S, M, F)$, where $N$ and $T$ are the sets of terminal and non-terminal symbols, respectively, $S$ is the start symbol, $M$ is the set of matrices, and $F$ is the set of productions that can be used in the appearance checking mode. We say that $G$ is in the *Z-binary normal form* if $N = N_1 \cup N_2 \cup \{S, Z, \#\}$, with these three sets being mutually disjoint, and the matrices in $M$ are in one of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$,
2. $(X \to Y, A \to x)$, with $X, Y \in N_1$, $A \in N_2$, $x \in (N_2 \cup T)^*$, $|x| \leq 2$,
3. $(X \to Y, A \to \#)$, with $X \in N_1$, $Y \in N_1 \cup \{Z\}$, $A \in N_2$,
4. $(Z \to \lambda)$.

Moreover, there is only one matrix of type 1, $F$ consists exactly of all rules $A \to \#$ appearing in matrices of type 3, and, if a sentential form generated by $G$ contains the symbol $Z$, then it is of the form $Zw$, for some $w \in (T \cup \{\#\})^*$ (that is, the appearance of $Z$ makes sure that, except for $Z$, all symbols are either terminal or the trap symbol $\#$). The (unique) matrix of type 4 is used only once, in the last step of a derivation. Finally, at most three non-terminal symbols are used in the appearance checking mode.

The following propositions are consequences of the results elaborated in [4], [6], and [7]:

**Proposition 3.** *For each language $L \in RE$ there is a matrix grammar with appearance checking $G$ in the Z-binary normal form such that $L = L(G)$.*

**Proposition 4.** *For each set $L \in N_0^{\beta}RE$ there is a matrix grammar with appearance checking $G$ in the Z-binary normal form such that $L(G)$ is a representation of $L$.*

In several of the following proofs we will start from matrix grammars in the Z-binary normal form. In order not to repeat the same notations, we consider the following representation (notations and labelling) for such grammars:

A matrix grammar with appearance checking in the Z-binary normal form is always given as $G = (N, T, S, M, F)$, with $N = N_1 \cup N_2 \cup \{S, Z, \#\}$, and with $n + 2$ matrices in $M$, injectively labelled with $m_0, m_1, \ldots, m_n, m_{n+1}$; the matrix (of type 1) $m_0 : (S \to X_{init}A_{init})$ is the initial one, with $X_{init}$ a given symbol from $N_1$ and $A_{init}$ a given symbol from $N_2$; the next $k$ matrices (of type 2) contain only rules without appearance checking, $m_i : (X \to Y, A \to x)$, $1 \leq i \leq k$, where $X, Y \in N_1$, and $A \in N_2$, $x \in (N_2 \cup T)^*$, $|x| \leq 2$; the next $n-k$ matrices (of type 3) have rules to be applied in the appearance checking mode, $m_i : (X \to Y, A \to \#)$, $k + 1 \leq i \leq n$, with $X \in N_1$, $Y \in N_1 \cup \{Z\}$, $A \in N_2$. Finally, $m_{n+1} : (Z \to \lambda)$ is the unique terminal matrix (of type 4).

Thus, when we will say "a matrix grammar in the Z-binary normal form, *with the standard notations/representation*" we will mean the notations and the conventions given here.

## 3 Membrane Systems with Symport/Antiport Rules

In this section we recall the definition of P systems with symport/antiport as introduced in [14]. After having recalled several results known so far for this model of P systems we show how some of these results can be improved.

### 3.1 P Systems with Symport/Antiport

We assume the reader to have some familiarity with membrane computing, so we do not recall the few basic notions from this area used here. For details we refer the interested reader to the monographs [2], [19] and to the P Systems Web Page [13].

We start from the biological observation that there are many cases where two chemicals pass at the same time through a membrane, with the help of each other, either in the same direction, or in opposite directions; in the first case we say that we have *symport*, in the second case we have *antiport* (we refer to [1] for details).

Mathematically, we can capture the idea of symport by considering 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. For antiport we consider rules of the form $(a, out; b, in)$, stating that $a$ exits and at the same time $b$ enters the membrane. Generalizing such kinds of rules, we can consider rules of the unrestricted forms $(x, in)$, $(x, out)$ (generalized symport) and $(x, out; y, in)$ (generalized antiport), where $x, y$ are non-empty strings representing multisets of objects, without any restriction on the length of these strings.

Based on rules of this types, in [14] we find *P systems with symport/antiport* as constructs

$$\Pi = (V, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m, i_o),$$

where:

1. $V$ is an alphabet (its elements are called *objects*);
2. $\mu$ is a membrane structure consisting of $m$ membranes, with the membranes (and hence the regions) injectively labelled with $1, 2, \ldots, m$; $m$ is called the *degree* of $\Pi$;
3. $w_i, 1 \leq i \leq m$, are strings over $V$ representing multisets of objects associated with the regions $1, 2, \ldots, m$ of $\mu$, present in the system at the beginning of a computation;
4. $E \subseteq V$ is the set of objects which are supposed to continuously appear in the environment in arbitrarily many copies;
5. $R_1, \ldots, R_m$ are finite sets of symport and antiport rules over the alphabet $V$ associated with the membranes $1, 2, \ldots, m$ of $\mu$;
6. $i_o$ is the label of an elementary membrane of $\mu$ (the *output membrane*).

For a symport rule $(x, in)$ or $(x, out)$, we say that $|x|$ is the *weight* (or *radius*) of the rule. The *weight* of an antiport rule $(x, out; y, in)$ is $\max\{|x|, |y|\}$, its *radius* is $(|x|, |y|)$.

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 $E$ (in the environment) are supposed to appear in arbitrarily many copies; since we only move objects from a membrane to another membrane and do not create new objects in the system, we need a supply of objects in order to compute with arbitrarily large multisets. The rules are used in the non-deterministic maximally parallel manner specific to P systems with symbol objects.

In this way, we obtain transitions between the configurations of the system. A configuration is described by the $m$-tuple of the multisets of objects present in the $m$ regions of the system, as well as the multiset of objects from $V \setminus E$ which were sent out of the system during the computation; it is important to keep track of such objects because they appear only in a finite number of copies in the initial configuration and can enter the system again. 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 $(w_1, \ldots, w_m, \lambda)$. A sequence of transitions is called a computation.

With any halting computation, we may associate an output of the system $\Pi$ represented by the numbers of objects from $V$ present in membrane $i_o$ in the halting configuration; these numbers of objects give rise to several variants of interpretation: In the original definition presented in [14], the total number of objects was considered the output computed by the system $\Pi$ and was denoted by $N(\Pi)$. Now, for $V = \{a_1, \ldots, a_n\}$ and every $\gamma$ with $\gamma \leq n$, let $V_\gamma = \{a_1, \ldots, a_\gamma\}$; then we may distinguish between these first $\gamma$ objects from $V$ and consider the set of vectors of numbers of objects from $V_\gamma$ (or equivalently, the set of multisets over $V_\gamma$) in a halting computation of the system $\Pi$ as its result, provided that any halting computation of $\Pi$ ends up with only objects from $V_\gamma$ in the output membrane; these sets of vectors (multisets) then are denoted by $N^\gamma(\Pi)$. The families of all sets $N(\Pi)$ and $N^\gamma(\Pi)$ computed by systems $\Pi$ of degree at most $m \geq 1$, using symport rules of weight at most $p$ and antiport rules of weight at most $q$, are denoted by $NP_m(sym_p, anti_q)$ and $N^\gamma P_m(sym_p, anti_q)$, respectively; when any of the parameters $m, p, q$ is not bounded, we replace it with $*$.

### 3.2 Previous Universality Results for P Systems with Symport/Antiport

P systems with symport/antiport rules were already considered in several papers, e.g., see [9], [10], [14], [15], [16], [20]. We now recall some of the universality results from these papers, without proofs. Observe that usually only sets of non-negative (positive) integers instead of the more general case of sets of vectors of non-negative (positive) integers are considered there.

First, we mention a universality result for P systems using both symport rules as well as antiport rules:

**Theorem 1.** $NP_2\,(sym_2, anti_2) = N_0^1RE$.

At the price of using more membranes, one can eliminate the antiport rules:

**Theorem 2.** $NP_5\,(sym_2, anti_0) = N_0^1RE$.

The number of membranes can be reduced at the expense of using symport rules of a larger weight (and no antiport rule). In fact, a trade-off relation seems to exist between the number of membranes and the weight of symport rules necessary for obtaining computational universality.

**Theorem 3.** $NP_3\,(sym_4, anti_0) = NP_2\,(sym_5, anti_0) = N_0^1RE$.

At first sight, antiport rules are a generalization of symport rules, because a rule $(u, out)$ can be transformed into $(u, out; d, in)$, where $d$ is a dummy object, and the same for rules $(u, in)$. Actually, this is not true, as it is clear that no system $\Pi$ using only antiport rules can compute both the number zero and any non-zero number. However, all recursively enumerable sets of positive numbers, i.e., $N_1^1RE$, can be computed by P systems using only antiport rules. Throughout the rest of the paper, we will not take into account this singleton set representing the number zero any more when dealing with families of sets computed by P systems using only antiport rules.

**Theorem 4.** $NP_3\,(sym_0, anti_2) = N_1^1RE$.

The number of membranes used in Theorem 2 for characterizing $N_0^1RE$ can be reduced if the symport rules have permitting or forbidding context conditions (promoting or inhibiting objects). A symport rule with a permitting condition is given in the form $(x, in)_a$, $(x, out)_a$, where $a$ is an object; this object should be present in membrane $i$ when a rule $(x, in)_a$, $(x, out)_a$ is applied (in the second case, $a$ should not be an element of $x$). A symport rule with a forbidding condition is given in the form $(x, in)_{\neg a}$, $(x, out)_{\neg a}$ and such a rule can be used only if object $a$ *is not present* in the membrane where the rule is applied.

The use of permitting (forbidding) conditions is indicated by replacing *sym* by *psym* (*fsym*, respectively) in the notation $NP_m\,(sym_p, anti_q)$.

**Theorem 5.** $NP_3\,(psym_2, anti_0) = NP_3\,(fsym_2, anti_0) = N_0^1RE$.

### 3.3 Improvements of Previous Results for P Systems with Symport/Antiport

We will now present the new results obtained in this paper. We first improve Theorem 3; we actually improve both relations in the theorem; en passant, we also extend the result from sets of non-negative integers to sets of vectors of non-negative integers:

**Theorem 6.** *For every* $\beta \geq 1$, $N^\beta P_2\,(sym_3, anti_0) = N_0^\beta RE$.

*Proof.* We only have to prove $N^{\beta}P_2\,(sym_3, anti_0) \supseteq N_0^{\beta}RE$. For that purpose, let us consider a matrix grammar with appearance checking $G = (N, T, S, M, F)$ in the Z-binary normal form with the standard notations/representation such that $T = \{a_i \mid 1 \leq i \leq \beta\}$; we now construct the following P system of degree 2 with symport rules of weight $\leq 3$:

$$
\begin{aligned}
\Pi &= (V, [_1[_2\,]_2]_1, w_1, \lambda, E, R_1, R_2, 2)\,,\\
V &= T \cup N_1 \cup N_2 \cup \{d_i, d_i', d_i'' \mid 1 \leq i \leq n\} \cup \{A' \mid A \in N_2\} \cup \{b, c, Z, \#\}\,,\\
w_1 &= X_{init}A_{init}bcd_1d_2 \ldots d_nd_1''d_2'' \ldots d_n'',\\
E &= N_1 \cup N_2 \cup T \cup \{d_i' \mid 1 \leq i \leq n\} \cup \{A' \mid A \in N_2\} \cup \{Z, \#\}\,,\\
R_1 &= \{(d_iXA, out)\,,\ (d_id_i'\alpha_1, in)\,,\ (d_i'd_i'', out)\,,\ (d_i''Y\alpha_2, in) \mid \text{ for}\\
&\qquad m_i : (X \to Y, A \to \alpha_1\alpha_2)\,,\ 1 \leq i \leq k, \text{ with}\\
&\qquad X, Y \in N_1,\ A \in N_2,\ \alpha_1, \alpha_2 \in N_2 \cup T \cup \{\lambda\}\}\\
&\quad \cup \{(d_iX, out)\,,\ (d_id_i'A', in)\,,\ (d_i'd_i'', out)\,,\ (d_i''YA', in) \mid \text{ for}\\
&\qquad m_i : (X \to Y, A \to \#)\,,\ k+1 \leq i \leq n, \text{ with}\\
&\qquad X, Y \in N_1 \cup \{Z\}\,,\ A \in N_2\}\\
&\quad \cup \{(A'Ab, out)\,,\ (A'A', out) \mid A \in N_2\} \cup \{(b\#, in)\,,\ (cZ, out)\}\,,\\
R_2 &= \{(a, in) \mid a \in T\} \cup \{(\#, in)\,,\ (\#, out)\,,\ (c, in)\,,\ (c, out)\}\,.
\end{aligned}
$$

We now show that every derivation in $G$ can be simulated by a halting computation in $\Pi$, the result showing up in membrane 2, and that the result of any halting computation in $\Pi$ corresponds to the result of a terminal derivation in $G$.

"$L\,(G) \subseteq N^{\beta}\,(\Pi)$"

Let us consider a successful derivation in $G$ generating some $w \in T^*$:

$$S \Longrightarrow X_{init}A_{init} \Longrightarrow \ldots \Longrightarrow Zw \Longrightarrow w.$$

We can simulate this derivation in $\Pi$ in the following way: Initially we already have $X_{init}A_{init}$ (i.e., we have already simulated the unique start matrix from $G$) in membrane 1; there we will simulate all the (other) matrices from $G$. Let us consider a matrix $m_i : (X \to Y, A \to \alpha_1\alpha_2)$ of type 2 from $G$, for which we have the following symport rules in $\Pi$:

1. $(d_iXA, out)$ checks that both $X$ and $A$ appear in membrane 1; using
2. $(d_id_i'\alpha_1, in)$ then $d_i$ will come back together with $\alpha_1$ (i.e., the first half of the right side of production $A \to \alpha_1\alpha_2$) as well as $d_i'$, which by
3. $(d_i'd_i'', out)$ is sent out together with $d_i''$, which by
4. $(d_i''Y\alpha_2, in)$ comes back into the skin membrane together with $Y$ and the second half of the right side of production $A \to \alpha_1\alpha_2$.

So after these four steps we have replaced $X$ by $Y$ and $A$ by $\alpha_1\alpha_2$, whereas all the other symbols have remained in the same regions as before; hence, the

matrix of type 2 was simulated correctly. Observe that $\alpha_1$ or $\alpha_2$ or even both may be the empty word $\lambda$.

For the simulation of a matrix $m_i : (X \to Y, A \to \#)$ of type 3 (i.e., those working in the appearance checking manner) we have the following symport rules in $\Pi$:

1. $(d_i X, out)$ starts the simulation by expelling $d_i$ together with $X$; by using
2. $(d_i d_i' A', in)$ then $d_i$ returns together with $d_i'$ and $A'$.
   $A'$ performs the appearance checking for $A$, i.e., if $A$ is present, then by using the rule $(A'Ab, out)$ both $A'$ and $A$ will be expelled together with $b$, which will return into the skin membrane together with the trap symbol # in the next step by the rule $(b\#, in)$. If, in the desired case, $A$ is not present in the system, then after the application of the rule
3. $(d_i' d_i'', out)$, the next rule
4. $(d_i'' Y A', in)$ allows $d_i''$ to return into the skin membrane together with $Y$ and $A'$; the rule
5. $(A'A', out)$ then finishes the correct simulation of the application of the matrix $m_i : (X \to Y, A \to \#)$ of type 3 from $G$.

If $A$ originally has not been present, we have correctly simulated the application of the matrix $m_i : (X \to Y, A \to \#)$ - $X$ has been replaced by $Y$, and the system has checked for the appearance of $A$. On the other hand, if $A$ had been present, the computation would have been "killed" by bringing in the trap symbol #, thus the computation would never end.

It is clear now that we can iterate these steps simulating any matrix of type 2 and type 3 from $G$ in $\Pi$. At the end of the simulation of a derivation in $G$, $Z$ leaves the system together with $c$, and no other rules can be applied any more (unless # is present). Hence, we have reached a halting computation with the result of the simulated derivation in the matrix grammar $G$ being stored in membrane 2, which proves $L(G) \subseteq N^{\beta}(\Pi)$.

"$L(G) \supseteq N^{\beta}(\Pi)$"

It is easy to see that we cannot simulate more than one matrix at one time because in membrane 1 we always have at most one non-terminal symbol from $N_1$ (initially we have $X_{init}$, and the matrices from the matrix grammar do not increase the number of the non-terminal symbols from $N_1$); while simulating a matrix we make sure that we bring in the non-terminal from $N_1$ at the end of the simulation step, so that we cannot start the simulation of another matrix before the previous steps have been completed.

Hence, at one specific moment of computation, only one matrix rule involving a non-terminal symbol from $N_1$ can be applied, meaning that at most one $d_i$ can be outside of the system at this moment, which implies that we will have at most one $d_i'$ inside the system and at most one $d_i''$ outside the system.

It is clear now that the simulation of matrices of type 2 from $G$ does not introduce "bad" symbols. Matrices of type 3 differ by the fact that if $A$ is present in the membrane when the first $A'$ is brought in (by $d_i'$), then they exit together

using $(AA'b, out)$ so that the trap symbol # enters the system by $(b\#, in)$. Even though later $d_i''$ brings in a second copy of $A'$ which will remain there and might create problems with further simulations, the computation will never stop because # will not exit the system, continuously moving between membranes 1 and 2.

The last observation we make is that, to reach a halting configuration, $c$ and # have to be absent from the system; $c$ is initially present and leaves the system only together with $Z$; but if $Z$ is present this means that the derivation in $G$ was terminal (except for possible occurrences of the trap symbol #), no other symbols from $N_1 \cup N_2$ are present. If # is present, then the system will not halt; yet if # is not present, then the derivation in $G$ is terminal and the computation in $\Pi$ halts. Hence, we have proved $N^{\beta}(\Pi) \subseteq L(G)$, too. □

**Corollary 1.** $NP_2(sym_3, anti_0) = N_0^1 RE$.

## 4 Communicating P Systems

We now consider another variant of P Systems, the so-called *communicating P systems* as defined in [24]. The difference between the previous model and this one is that now we do not use symport/antiport rules, but purely communicating rules of quite similar types.

### 4.1 Definition of Communicating P Systems

A *communicating P system* is a P system

$$\Pi = (V, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m, i_o),$$

where every $R_i$ contains only the following types of rules: $a \to a_\tau$, $ab \to a_{\tau_1} b_{\tau_2}$, $ab \to a_{\tau_1} b_{\tau_2} c_{come}$, where $a, b, c \in V$, $\tau, \tau_1, \tau_2 \in \{here, in, out\}$. The rest of the "ingredients" of a communicating P system are the same as in the variant discussed in section 3.

Let us explain now the use of these new types of rules. The objects follow their target indication (one can easily see that we do not create or destroy objects, we just move them around). If the target says "here", then that object will remain in the same region (we will omit this target indication from now on; so the objects that remain in the same region will not have any target indication). If the target indication is "in", then the object goes in a "directly enclosed" region; to be able to apply this rule there must be at least one membrane inside. If the target says "out", then that object is expelled from the current region into the surrounding one, or into the environment if the rule is applied in the skin membrane. Finally the target indication "come" can only be used if a rule of the form $ab \to a_{\tau_1} b_{\tau_2} c_{come}$ appears in the skin membrane, and by applying this rule, the object with that target "*come*" is "imported" through the skin membrane from the environment (and thus will become an element of the skin membrane).

As before, such a system works in a non-deterministic maximally parallel manner until no rules can be applied any more. When the system halts, the result of the computation is found in the output membrane $i_o$ defined in $\Pi$.

### 4.2 The Universality of Communicating P Systems

Our first construction improves a result from [24] where the number of membranes was not fixed; as in [24], the proof is based on results for register machines (see Minsky [12]) using Proposition 1.

**Theorem 7.** *Any partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$ can be computed by a communicating P system with only one membrane.*

*Proof.* Let $m = \max\{\alpha, \beta\} + 2$. According to Proposition 1 there exists a deterministic $m$-register machine $M = (m, P, 1, n)$ computing $f$ in such a way that, when starting with $(n_1, ..., n_\alpha) \in \mathbf{N}_0^\alpha$ in registers 1 to $\alpha$, $M$ has computed $f(n_1, ..., n_\alpha) = (r_1, ..., r_\beta)$ if it halts in the final label $n$ with registers 1 to $\beta$ containing $r_1$ to $r_\beta$ (and with all other registers being empty); if the final label cannot be reached, $f(n_1, ..., n_\alpha)$ remains undefined. We now construct a communicating P system simulating this particular $m$-register machine with $n$ instructions (for the sake of better readability, we attach the target indications "*out*" and "*come*" as second index to indexed variables, e.g., we write $a_{r,come}$ instead of $a_{r_{come}}$):

$$
\begin{aligned}
\Pi &= (V, [_1\ ]_1, w_1, E, R_1 \cup R_2 \cup R_3 \cup R_4, 1),\\
V &= \{a_i \mid 1 \le i \le m\} \cup \{c, d, \#, n, n'\},\\
&\quad \cup \{j, j', j'' \mid j : (A(r), k, k) \in P, 1 \le j < n, 1 \le r \le m, 1 \le k \le n\}\\
&\quad \cup \left\{j, j', j'', j''', j^{iv} \mid j : (S(r), k, l) \in P, 1 \le j < n, 1 \le r \le m, 1 \le k, l \le n\right\},\\
w_1 &= 1cd,\\
E &= V,\\
R_1 &= \{jc \to jcj'_{come},\ j'd \to j'_{out}d_{out}\#_{come},\ jj' \to j_{out}j'j''_{come},\\
&\qquad j'j'' \to j'_{out}j''a_{r,come},\ j''c \to j''_{out}ck_{come} \mid\\
&\qquad 1 \le j < n,\ j : (A(r), k, k) \in P,\ 1 \le r \le m,\ 1 \le k \le n\},\\
R_2 &= \{ja_r \to j_{out}a_{r,out}k_{come},\ jc \to jcj'_{come},\ j'd \to j'_{out}d_{out}\#_{come},\\
&\qquad jj' \to j_{out}j'j''_{come},\ j'j'' \to j'_{out}j''j'''_{come},\ j''c \to j''_{out}cj^{iv}_{come},\\
&\qquad j'''a_r \to j'''_{out}a_{r,out}\#_{come},\ j'''j^{iv} \to j'''_{out}j^{iv}_{out}l_{come} \mid\\
&\qquad 1 \le j < n,\ j : (S(r), k, l) \in P,\ 1 \le r \le m,\ 1 \le k, l \le n\},\\
R_3 &= \{nc \to n_{out}c_{out}n'_{come},\ n'd \to n'_{out}d_{out}\},\\
R_4 &= \{\# \to \#\}.
\end{aligned}
$$

Let us now see how the communicating P system $\Pi$ simulates the deterministic register machine $M$.

With the rules from $R_1$ we simulate all the ADD instructions from the given register machine $M$: For such a rule $j : (A(r), k, k)$ we first bring in $j'$ into

the system using $jc \to jcj'_{come}$; this rule could be used again at the next step bringing another copy of $j'$ into the system, but then, the previous copy of $j'$ would have to use the rule $j'd \to j'_{out}d_{out}\#_{come}$ and the computation would never finish because of the presence of the trap symbol #. So, it is clear that the rule $jc \to jcj'_{come}$ should be applied only once if we want to obtain a halting computation. In the next step $j$ and $j'$ together use $jj' \to j_{out}j'j''_{come}$, which brings in a copy of $j''$ at the same time when $j$ is expelled. If at this moment $j'$ uses the rule $j'd \to j'_{out}d_{out}\#_{come}$, we again "loose" the computation, so $j'$ and $j''$ have to continue together using the rule $j'j'' \to j'_{out}j''a_{r,come}$, which sends out $j'$ and at the same time brings in $a_r$; in this way we increase register $r$. We have to finish the simulation of $j : (A(r), k, k)$ by replacing $j''$ with $k$ using $j''c \to j''_{out}ck_{come}$.

A SUBTRACT instruction $j : (S(r), k, l)$ is simulated by using rules from $R_2$: If we can use the rule $ja_r \to j_{out}a_{r,out}k_{come}$, then this means that the contents of register $r$ was not zero, so the rule was simulated correctly in this case. Suppose that instead we use the rule $jc \to jcj'_{come}$ (which starts the simulation of the case when register $r$ is empty); as before, if the newly introduced symbol $j'$ is used by the rule $j'd \to j'_{out}d_{out}\#_{come}$, then the computation is lost, so to get a result in a halting computation $j'$ has to be used together with $j$ by the rule $jj' \to j_{out}j'j''_{come}$, and again – to avoid a "disaster" – $j'$ has to be used together with $j''$ by the rule $j'j'' \to j'_{out}j''j'''_{come}$. At this moment we have $j''$, $j'''$, $c$, $d$ in our system together with the symbols $a_i$ representing the contents of the registers. Now $j'''$ will just check whether register $r$ is really empty; if this is not true, then the rule $j'''a_r \to j'''_{out}a_{r,out}\#_{come}$ can (and will) be applied thus "killing" the computation. At the same time $j''$ brings in $j^{iv}$ with the rule $j''c \to j''_{out}cj^{iv}_{come}$; the simulation of the SUBTRACT instruction $j : (S(r), k, l)$ correctly can be finished by the application of the rule $j'''j^{iv} \to j'''_{out}j^{iv}_{out}l_{come}$.

One can easily see that we can iterate these steps since we always return to a configuration in which we have one instruction label, $c$, $d$ as well as the symbols $a_i$ representing the contents of the registers in the skin membrane.

At the end we simulate the HALT instruction $n : HALT$ using the "clean-up" rules $nc \to n_{out}c_{out}n'_{come}$ and $n'd \to n'_{out}d_{out}$ from $R_3$; thus, only the symbols $a_i$ representing the contents of the output registers finally remain in the skin membrane. □

*Remark 1.* The following theorem directly follows from Proposition 2 as Theorem 7 followed from Proposition 1; for the simulation of a non-deterministic ADD instruction $j : (A(r), k, l)$, we only have to add an additional rule $j''c \to j''_{out}cl_{come}$ for the second label $l$ in the set $R_1$.

We will give another proof of this universality result based on the simulation of matrix grammars with appearance checking (in the Z-binary normal form), because the constructions given in the proof seem to be quite interesting for themselves when compared with other proofs in the area of membrane systems:

**Theorem 8.** *Any set $L \in N_0^{\beta}RE$ can be computed by a communicating P system with only one membrane.*

*Proof.* Let us consider the terminal alphabet $T = \{a_i \mid 1 \leq i \leq \beta\}$ and a matrix grammar with appearance checking $G = (N, T, S, M, F)$ in the Z-binary normal form with the standard notations/representation which generates $L$; we now construct the following communicating P system of degree 1 which computes (generates) $L$:

$$
\begin{aligned}
\Pi &= \{V, [_1\ ]_1, w_1, E, R_1 \cup R_2 \cup R_3 \cup R_4, 1\}, \\
V &= T \cup N_1 \cup N_2 \cup \{c, d, \#, Z\} \cup \{X', X'', \overline{X} \mid X \in N_1 \cup \{Z\}\} \\
&\quad \cup \{A'_Y, A''_Y, A'''_Y \mid A \in N_2,\ Y \in N_1 \cup \{Z\}\} \cup \{\overline{A} \mid A \in N_2\}, \\
w_1 &= X_{init} A_{init} cd, \\
E &= V, \\
R_1 &= \{Xc \to Xc X'_{come},\ X'd \to X'_{out} d_{out} \#_{come},\ XX' \to X_{out} X' X''_{come}, \\
&\quad X'X'' \to X'_{out} X'' \alpha_{1,come},\ X''c \to X''_{out} c A'_{Y,come},\ A'_Y c \to A'_Y c A''_{Y,come}, \\
&\quad A''_Y d \to A''_{Y,out} d_{out} \#_{come},\ A'_Y A''_Y \to A'_{Y,out} A''_Y A'''_{Y,come}, \\
&\quad A''_Y A'''_Y \to A''_{Y,out} A'''_Y \alpha_{2,come},\ A'''_Y A \to A'''_{Y,out} A_{out} Y_{come} \mid \\
&\quad m_i : (X \to Y, A \to \alpha_1 \alpha_2),\ 1 \leq i \leq k, \\
&\quad X, Y \in N_1,\ A \in N_2,\ \alpha_1, \alpha_2 \in N_2 \cup T \cup \{\lambda\}\} \\
&\quad \cup \{A'''_Y c \to A'''_{Y,out} c_{out} \#_{come} \mid A \in N_2, Y \in N_1\}, \\
R_2 &= \{Xc \to Xc X'_{come},\ X'd \to X'_{out} d_{out} \#_{come}, \\
&\quad XX' \to X_{out} X' X''_{come},\ X'X'' \to X'_{out} X'' \overline{A}_{come}, \\
&\quad \overline{A}A \to \overline{A}_{out} A_{out} \#_{come},\ X''c \to X''_{out} c \overline{Y}_{come}, \\
&\quad \overline{YA} \to \overline{Y}_{out} \overline{A}_{out} Y_{come} \mid m_i : (X \to Y, A \to \#), \\
&\quad k+1 \leq i \leq n,\ X \in N_1, Y \in N_1 \cup \{Z\}, A \in N_2\}, \\
R_3 &= \{Zc \to Z_{out} c_{out} Z'_{come},\ Z'd \to Z'_{out} d_{out}\}, \\
R_4 &= \{\# \to \#\}.
\end{aligned}
$$

The communicating P system $\Pi$ works in the following way.

Initially we have $X_{init} A_{init}$ in the skin membrane, and thus we have already simulated the unique rule of type 1 from the matrix grammar $G$.

For a matrix $(X \to Y, A \to \alpha_1 \alpha_2)$ of type 2 from $G$ we perform the corresponding rules from $R_1$, and for a matrix $(X \to Y, A \to \#)$ of type 3 from $G$ we perform the corresponding rules from $R_2$.

We will not go into the details of the construction since we used similar ideas as in the previous proof, so we leave these verifications to the reader. On the other hand, some small observations seem to be necessary. We always have rules that check the correctness of the simulation, and in case the simulation does not follow a correct path, the trap symbol # is brought into the system and never leaves it again, hence, the system will never halt, because the rule $\# \to \#$ from $R_4$ is always applicable to the trap symbol #.

The simulation of a matrix $(X \to Y, A \to \alpha_1 \alpha_2)$ has the following main steps: We replace $X$ by $\alpha_1$ and $A'_Y$, then $A'_Y$ is replaced by $A'''_Y$ and $\alpha_2$; $A'''_Y$ exits together with $A$ and brings in $Y$. Observe that $\alpha_1$ or $\alpha_2$ or even both may

be the empty word $\lambda$; in these cases, we have to replace the rule $X'X'' \to X'_{out}X''\alpha_{1,come}$ by $X'X'' \to X'_{out}X''$ or/and the rule $A''_Y A'''_Y \to A''_{Y,out}A'''_Y\alpha_{2,come}$ by $A''_Y A'''_Y \to A''_{Y,out}A'''_Y$ in $R_1$.

The simulation of a matrix $(X \to Y, A \to \#)$ is performed by the following steps: We replace $X$ by $\overline{A}$ and $\overline{Y}$; since $\overline{A}$ comes before $\overline{Y}$, it can check whether $A$ is present in the system by the rule $\overline{A}A \to \overline{A}_{out}A_{out}\#_{come}$; if this is not the case, then $\overline{A}$ can exit together with $\overline{Y}$ bringing in $Y$.

We finish the computation by using the rules from $R_3$ because in that moment (since $Z$ appeared in the system) we have only terminal symbols from $T$ in the skin membrane, the special symbols $c, d$ and, maybe, some trap symbols #. If # is present, then the computation will never finish (because of the rule $\# \to \#$ in $R_4$); on the other hand, if # is not present, then the system halts after the application of the two rules from $R_3$.

A computation in the communicating P system $\Pi$ halts (yielding the same result as the corresponding derivation in $G$) if and only if the applications of the matrices from $G$ in a terminal derivation in $G$ have been simulated correctly (this is ensured by the final appearance of $Z$ and from the definition of the Z-binary normal form). □

### 4.3 Interpreting Communicating P Systems with Only One Membrane as P Systems with Symport/Antiport

Returning to the proof of Theorem 7, we realize the surprising fact that communicating P systems with the simplest membrane structure allow for an immediate interpretation as P systems with symport/antiport, because the skin membrane includes no inner membrane, so that the target indication "*in*" cannot occur in the rules of such a simple communicating P system.

The following table shows how the remaining variants of rules in a communicating P system with only one membrane can be written as symport/antiport rules:

| communicating P system | P system with symport/antiport |
|---|---|
| $a \to a$ | $(a, out; a, in)$ |
| $a \to a_{out}$ | $(a, out)$ |
| $ab \to ab$ | $(ab, out; ab, in)$ |
| $ab \to a_{out}b$ | $(ab, out; b, in)$ |
| $ab \to a_{out}b_{out}$ | $(ab, out)$ |
| $ab \to abc_{come}$ | $(ab, out; abc, in)$ |
| $ab \to a_{out}bc_{come}$ | $(ab, out; bc, in)$ |
| $ab \to a_{out}b_{out}c_{come}$ | $(ab, out; c, in)$ |

In order to obtain an optimal improvement of Theorem 1 and Theorem 4 we need some more technical details as they are elaborated in the proof of the following theorem:

**Theorem 9.** *Any partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$ can be computed by a P system with symport/antiport rules that consists of the simplest membrane*

*structure and only uses antiport rules of the form* $(x, out; y, in)$ *with the radius of the rules being* $(2,1)$ *or* $(1,2)$, *respectively, and only one single symport rule of radius* 1.

*Proof.* Translating the construction of the communicating P system in the proof of Theorem 7 according to the table given above and taking into account some further technical details explained below, we immediately get the following P system with symport/antiport:

$$
\begin{aligned}
\Pi &= (V, [_1\ ]_1, w_1, E, R_1 \cup R_2 \cup R_3 \cup R_4, 1), \\
V &= \{a_i \mid 1 \le i \le m\} \cup \{c, d, \#, \#', \#'', n, n', n''\}, \\
&\quad \cup \{j, j', j'' \mid j : (A(r), k, k) \in P, 1 \le j < n, 1 \le r \le m, 1 \le k \le n\} \\
&\quad \cup \left\{j, j', j'', j''', j^{iv} \mid j : (S(r), k, l) \in P, 1 \le j < n, 1 \le r \le m, 1 \le k, l \le n\right\}, \\
w_1 &= 1cd, \\
E &= V, \\
R_1 &= \{(j, out; jj', in),\ (j'd, out; \#, in),\ (jj', out; j'j'', in), \\
&\qquad (j'j'', out; j''a_r),\ (j''c, out; ck, in) \mid \\
&\qquad 1 \le j < n,\ j : (A(r), k, k) \in P,\ 1 \le r \le m,\ 1 \le k \le n\}, \\
R_2 &= \{(ja_r, out; k, in),\ (j, out; jj', in),\ (j'd, out; \#, in), \\
&\qquad (jj', out; j'j'', in),\ (j'j'', out; j''j'''),\ \left(j''c, out; cj^{iv}, in\right), \\
&\qquad (j'''a_r, out; \#, in),\ \left(j'''j^{iv}, out; l, in\right) \mid \\
&\qquad 1 \le j < n,\ j : (S(r), k, l) \in P,\ 1 \le r \le m,\ 1 \le k, l \le n\}, \\
R_3 &= \{(nc, out; n', in),\ (n'd, out; n'', in),\ (n'', out)\}, \\
R_4 &= \{(\#, out; \#'\#'', in), (\#'\#'', out; \#, in)\}.
\end{aligned}
$$

So far, we have taken into account the following changes in order to obtain the desired constraints for the P system with symport/antiport:

- The rules $jc \to jcj'_{come}$, $1 \le j < n$, were not replaced by the antiport rules $(jc, out; jcj', in)$ with weight 3, but instead could simply be replaced by the antiport rules $(j, out; jj', in)$ with radius $(1,2)$, because the symbol $c$ was only acting as a sort of "dummy catalyst" just in order to obey to the special form of rules in communicating P systems.
- The rule $n'd \to n'_{out}d_{out}$ was not replaced by the symport rule $(n'd, out)$ with weight 2, but instead by the antiport rule $(n'd, out; n'', in)$ with radius $(2,1)$ and the (single) symport rule $(n'', out)$ with weight 1.
- The rule $\# \to \#$ was not replaced by the antiport rule $(\#, out; \#, in)$ with radius $(1,1)$, but instead by the two antiport rules $(\#, out; \#'\#'', in)$ and $(\#'\#'', out; \#, in)$ with radius $(1,2)$ and $(2,1)$, respectively.

The only rules not yet obeying to the desired constraints are of the form $(ab, out; bc, in)$ with radius $(2,2)$. If we considered only the weight of the rules, we would already be finished. Yet each of these rules can be replaced by the two

antiport rules $(ab, out; x_{bc}, in)$ and $(x_{bc}, out; bc, in)$ using a new symbol $x_{bc}$; these antiport rules now have the desired radius $(2,1)$ and $(1,2)$, respectively. In that way, from the P system with symport/antiport $\Pi$ we obtain a P system with symport/antiport $\Pi'$ obeying to all the constraints required in the theorem. The remaining details of the proof are obvious and therefore left to the reader. □

We should like to mention that due to the special constraints for the rules in communicating P systems, the P system with symport/antiport constructed in the preceding theorem is not as simple as it could be, e.g., for simulating the ADD instruction $j : (A(r), k, k)$ we would only need the single antiport rule $(j, out; a_r k, in)$ in $R_1$. The task to reduce the number of rules in $R_2$ is left to the reader.

**Theorem 10.** *Any set $L \in N_0^{\beta} RE$ can be computed by a P system with symport/antiport rules that consists of the simplest membrane structure and only uses antiport rules of the form $(x, out; y, in)$ with the radius of the rules being $(2,1)$ or $(1,2)$, respectively, and only one single symport rule of radius 1.*

*Proof.* The result directly follows from Theorem 9 and Remark 1. □

The following result is a direct consequence of the preceding theorem and considerably improves Theorem 1:

**Corollary 2.** $NP_1(sym_1, anti_2) = N^1P_1(sym_1, anti_2) = N_0^1 RE$.

If we want to avoid the symport rule used in the last step of a computation of the P system with symport/antiport constructed in the proof of Theorem 9, we have to pay the price that at least one terminal symbol appears as the result of the computation, e.g., we may replace the rules $(n'd, out; n'', in)$ and $(n'', out)$ by the antiport rule $(n'd, out; a_1, in)$ thus avoiding the symport rule $(n'', out)$, which proves the following result:

**Theorem 11.** *Any partial recursive function $f : \mathbf{N}_0^{\alpha} \to \mathbf{N}_1 \times \mathbf{N}_0^{\beta-1}$ can be computed by a P system with antiport rules that consists of the simplest membrane structure and only uses antiport rules of the form $(x, out; y, in)$ with the radius of the rules being $(2,1)$ or $(1,2)$, respectively.*

As an immediate consequence of the preceding theorem we obtain the following optimal improvement of Theorem 4:

**Corollary 3.** $NP_1(sym_0, anti_2) = N^1P_1(sym_0, anti_2) = N_1^1 RE$.

The P systems with antiport rules considered in Corollary 3 are not only optimal with respect to the number of membranes but also with respect to the radius of the rules used in the system: Using only rules with radius $(1,1)$ and $(2,1)$ in the simplest membrane structure we only get finite sets of non-negative integers. If we only allow rules with radius $(1,1)$ and $(1,2)$ we at most get sets of non-negative integers representing monotonic one-letter languages.

We finally want to point out that, independently, similar improvements for P systems with symport/antiport as established in Corollaries 2 and 3 were achieved in [5] and [8], too.

## 5 Conclusions

We succeeded in improving some previous results for P systems with symport/antiport: For obtaining universal computational power for *P systems with symport* rules only, we could decrease the number of membranes to two and the upper bound for the number of symbols needed in the symport rules to three. It remains as an interesting open question whether the results for P systems with symport obtained so far still can be improved (observe that there seems to be a trade-off between the number of membranes needed and the number of symbols used together in one symport rule).

For the quite similar model of *communicating P systems* we even obtained an optimal universality result with respect to the number of membranes – only the simplest membrane structure consisting of the skin membrane only is already sufficient to obtain universal computational power. Moreover, by showing how to interpret communicating P systems with only one membrane as P systems with symport/antiport, we easily obtained optimal universality results for P systems with symport/antiport.

**Acknowledgements**

The research of Andrei Păun was supported by the Natural Sciences and Engineering Research Council of Canada grant OGP0041630 and a graduate scholarship.

## References

1. Alberts, B., et al.: Essential Cell Biology. An Introduction to the Molecular Biology of the Cell. Garland Publ. Inc., New York, London (1998)
2. Calude, C.S., Păun, Gh.: Computing with Cells and Atoms. Taylor & Francis, London (2001)
3. Dassow, J., Păun, Gh.: Regulated Rewriting in Formal Language Theory. Springer-Verlag, Berlin (1989)
4. Freund, R., Oswald, M.: GP Systems with Forbidding Context. Fundamenta Informaticae **49**, 1-3 (2002) 81–102
5. Freund, R., Oswald, M.: P Systems with Activated/Prohibited Membrane Channels, *this volume*
6. Freund, R., Păun, Gh.: On the Number of Non-terminals in Graph-controlled, Programmed, and Matrix Grammars. In: Margenstern, M., Rogozhin, Y. (eds.): Proc. Conf. Universal Machines and Computations, Chişinău (2001). Springer-Verlag, Berlin (2001)
7. Freund, R., Păun, Gh.: From Regulated Rewriting to Computing with Membranes: Collapsing Hierarchies; *to appear in TCS*
8. Frisco, P., Hoogeboom, H.J.: Simulating Counter Automata by P Systems with Symport/Antiport. In: [21] 237–248
9. Martin-Vide, C., Păun, A., Păun, Gh.: On the Power of P Systems with Symport Rules. J. Universal Computer Sci. **8** 2 (2002) 317–331

10. Martin-Vide, C., Păun, A., Păun, Gh., Rozenberg, G.: Membrane Systems with Coupled Transport: Universality and Normal Forms. Fundamenta Informaticae **49**, 1-3 (2002) 1–15
11. Martin-Vide, C., Păun, Gh.: Elements of Formal Language Theory for Membrane Computing. Technical Report 21/01 of the Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona (2001)
12. Minsky, M.L.: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, New Jersey (1967)
13. The P Systems Web Page: `http://psystems.disco.unimib.it/`
14. Păun, A., Păun, Gh.: The Power of Communication: P Systems with Symport/Antiport. New Generation Computing **20,** 3 (2002) 295–306
15. Păun, A., Păun, Gh., Rodriguez-Paton, A.: Further Remarks on P Systems with Symport Rules. Ann. Univ. Al.I. Cuza, Iaşi **10** (2001) 3–18
16. Păun, A., Păun, Gh., Rozenberg, G.: Computing by Communication in Networks of Membranes. International Journal of Foundations of Computer Science (2002) *accepted*
17. Păun, Gh.: Computing with Membranes. Journal of Computer and System Sciences **61,** 1 (2000) 108–143, and TUCS Research Report 208 (1998) (`http://www.tucs.fi`)
18. Păun, Gh.: Computing with Membranes: An Introduction. Bulletin EATCS **67** (1999) 139–152
19. Păun, Gh.: Membrane Computing: An Introduction. Springer-Verlag, Berlin (2002)
20. Păun, Gh., Perez-Jimenez, M., Sancho-Caparrini, F.: On the Reachability Problem for P Systems with Symport/Antiport. 10th Intern. Conf. on Automata and Formal Languages, Debrecen, 2002
21. Păun, Gh., Zandron, C. (eds.): Pre-Proceedings of Workshop on Membrane Computing (WMC-CdeA2002), Curtea de Argeş, Romania (2002)
22. Salomaa, A., Rozenberg, G. (eds.): Handbook of Formal Languages. Springer-Verlag, Berlin (1997)
23. Sosík, P.: P Systems Versus Register Machines: Two Universality Proofs. In: [21] 371–382
24. Sosík, P., Matysek, J.: Membrane Computing: When Communication is Enough. In: Calude, C.S., Dinneen, M.J., Peper, F. (eds.): Unconventional Models of Computation 2002, LNCS 2509, Springer-Verlag, Berlin (2002) 264–275

# Simulating Counter Automata by P Systems with Symport/Antiport*

Pierluigi Frisco and Hendrik Jan Hoogeboom

Institute for Advanced Computer Science
Universiteit Leiden, The Netherlands
http://www.liacs.nl/CS/TCS/

**Abstract.** The complexity, expressed in number of membranes and weight of rules, of P systems with symport/antiport generating recursively enumerable sets is reduced if counter automata instead of matrix grammars are simulated. We consider both subsets of $\mathbb{N}$ obtained by counting objects in a designated membrane, and string languages obtained by following the traces of a designated object.

## 1 Introduction

Lately several computability models inspired by direct observation of living systems have been proposed. Cells with their biochemical processes are studied as computational devices. The complex structure of the cell may be modelled as a set of (nested) compartments delimited by membranes, each compartment containing objects that may interact between themselves or pass to another compartment.

These reflections brought forth the definition of a model called *membrane systems* (and also *P systems*) introduced in [Pă00a] and later object of several investigations (see the bibliography [BibP]). A good tutorial overview into the field can be found in either one of two papers [MP01,PR02]. A book [Pă02b] covering the field of membrane computing has recently been published.

One of the variants of the original model was introduced in [PP02] under the name of membrane systems *with symport/antiport*. In this variant the only available operation is the synchronized movement of objects. Specific groups of objects may pass together through a membrane either in the same or in opposite direction. In the former case we refer to *symport*, in the latter to *antiport*.

These quite simple systems compute all recursively enumerable sets; the result has been obtained in [PP02], and then improved in complexity (reducing the number of membranes and the number of objects that are transported at the same time) in [M$^+$02a,M$^+$02b]. In these papers matrix grammars with appearance checking were considered as reference model for generating recursively enumerable sets, and their computations were simulated by P systems to obtain the completeness results.

* Work partially supported by contribution of EU commission under The Fifth Framework Programme, project "MolCoNet" IST-2001-32008

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 288–301, 2003.

In this paper we show that the simulation of counter automata (or register machines) instead of matrix grammars reduces the complexity of these systems further more. It was demonstrated in [FP01] that the simulation of counter automata brings down the number of nonterminals used in matrix grammars. The authors of [FP01] argue that this is "of crucial importance in obtaining universal P-sytems with a small number of membranes". We share their enthusiasm for counter automata in this context, but avoid the detour via matrix grammars. Regarding our own results, the counter automata approach is already present in [Ho02], where it is applied to P systems with *carriers* another model based on pure communication, i.e., where the objects in the system can only be moved around.

In Section 2 we provide an informal description of P systems with symport/antiport. We also fix the notion of counter automaton that we use in this paper. Section 3 deals with the basic complexity of P systems, the number of membranes. Indeed, we show that a single membrane suffices to obtain computational completeness. Then, in Section 4 we focus on systems where only symports are allowed. It turns out that moving four objects at a time suffices to obtain completeness, and only two membranes. Systems where symports are further reduced to move pairs of objects are considered in Section 5. Our result shows that four membranes are enough here. This bound could only be achieved previously for rules with additional control. Finally, we apply our constructions in Section 6 to P systems where strings are generated by following the movements of a designated object called the traveller. It seems we are the first to characterize recursively enumerable languages for these P systems without the help of additional control.

This paper was presented at the Workshop on Membrane Computing, Curtea de Argeş Romania, 2002 [PZ02]. There we have learned that some of our results have been obtained independently by others, and we have noted that using counter automata (or register machines) has become a standard practice in connection with P systems with unstructured objects. We have reacted to these developments by placing appropriate footnotes and by updating our final section.

## 2 Preliminaries

We assume the reader to have familiarity with basic concepts of formal language theory [HU79], and in particular with the topic of membrane systems with symport/antiport [PP02,M$^+$02a,M$^+$02b]. In this section we recall particular aspects relevant to our presentation.

We use $\mathbb{N}\cdot\mathsf{RE}$ to denote the family of recursively enumerable sets of natural numbers. For $k \in \mathbb{N}$, $\mathbb{N}_k\cdot\mathsf{RE}$ equals the family of recursively enumerable sets with elements greater or equal to $k$, $\{L \in \mathbb{N}\cdot\mathsf{RE} \mid \{0,\ldots,k-1\} \cap L = \varnothing\}$, or equivalently, $\{k+L \mid L \in \mathbb{N}\cdot\mathsf{RE}\}$, where $k+L = \{k+n \mid n \in L\}$. From the point of computational completeness, the families $\mathbb{N}\cdot\mathsf{RE}$ and $\mathbb{N}_k\cdot\mathsf{RE}$ are equivalent, as

a Turing machine can make the translation, but here we inherit the language definition of P systems and make this distinction.

**P Systems with Symport/Antiport.** A *membrane system*, or P system [Pă00a], is a computational device consiting of a set of hierarchically nested membranes. Within each membrane there may be objects evolving and moving to neighbouring membranes following rules specified for the particular membrane. Outside the outer membrane, the environment is subject to its own set of rules. Recent overviews [MP01,PR02], and –above all– the monograph [Pă02b], are a good source for motivation and for a tutorial on the various categories of P systems.

In P systems with symport/antiport as introduced in [PP02] computation is restricted to the synchronous movement of objects from one membrane into another, see [Pă02b, Chapter 4.1]. This means that the system contains unstructured objects that are not rewritten or changed in any other way.

A configuration of the system is given by a finite multiset for each of the compartments; each membrane contains a finite number of objects, whereas the objects initially present in the environment are assumed to have infinite (unbounded) supply. Rules (associated to the membranes) are of one of the following forms, where $x$ and $y$ are strings of objects (representing multisets in the obvious way):

- $(x, \text{in})$, multiset $x$ moves *into* the membrane from the compartment (membrane or environment) surrounding it;
- $(x, \text{out})$, multiset $x$ moves *out from* the membrane to the compartment surrounding it;
- $(x, \text{in}; y, \text{out})$, multiset $x$ moves into the membrane from the compartment surrounding it, while at the same time multiset $y$ moves into the other direction.

The first two forms $(x, \text{in})$ and $(x, \text{out})$ are called *symport* rules, the latter form $(x, \text{in}; y, \text{out})$ is called *antiport*. The *weight* of a rule is given by $|x|$ for symport, and $\max\{|x|, |y|\}$ for antiport.

Computational steps consist of the application of a multiset of these rules, under the usual requirement of maximal parallelism: such a multiset cannot be carried out if there is the possibility of performing a step that involves a strictly larger multiset of rules. A computation is successful if it starts in the initial configuration (specified as a multiset of objects for each membrane, and an infinite supply of objects in the environment) and if it ends in a halting configuration, i.e., a configuration where no rule is applicable. The result of a successful computation is the number of objects present in a designated membrane, the output membrane. By convention this membrane is elementary, i.e., it does not contain any membranes. As always, the subset of $\mathbb{N}$ specified by the system (its 'language') is the set of numbers resulting from successful computations.

We use $\mathbb{N}\cdot\mathsf{PP}_m(\text{sym}_j, \text{anti}_k)$ to denote the family defined by P systems with symport/antiport having at most $m$ membranes, symport of weight at most $j$, and antiport of weight at most $k$.

**Counter Automata.** A *counter automaton* is a finite state device equipped with additional external storage in the form of one or more counters. Each of these counters holds a natural number which may be incremented, decremented (when positive), and tested for zero. It is shown by Minsky [Mi61] that the counter automaton can simulate the Turing machine when it is equipped with (at least) two counters, see also [HU79, Theorem 7.9].

Here we use counter automata in the context of P systems generating (sets of) numbers, and we replace the input tape by an additional *output counter*, i.e., we syntactically replace reading input symbols by adding to the distinguished output counter; the output counter is never decremented nor tested for zero.

Formally, the transitions of the counter automaton are of the form $(p \to q, \iota)$, where $p$ and $q$ are states, and the instruction $\iota$ is either $\varepsilon$ or one of the three instructions $+A, -A, A = 0$ for any counter $A$ used by the automaton. In state $p$ the automaton may use this instruction changing state to $q$ and performing $\iota$ to the counters – where for $\iota = \varepsilon$ no counter is changed; for $+A$ counter $A$ is incremented by one; for $-A$ counter $A$ is decremented by one, defined only if the counter has a positive value; and for $A = 0$ counter $A$ is tested for zero, an instruction defined only if the counter has value zero.

A computation of the counter automaton is successful if starting from the initial state with empty counters it reaches the final state. The result of this computation is the number represented by the value of the output counter.

Without loss of generality we make some technical assumptions. We require that the final state can only be reached when all counters are zero (with the exception of the output counter, of course). This means that the counter automaton explicitly empties its counters before accepting. Additionally, we assume that the final state has no outgoing transitions.

**Conflicting Counters.** We use here a paradigm to implement the zero tests of the counter automaton, in the simulation by a P system. For each counter $A$ we introduce a corresponding counter, say $\bar{A}$, and we require that the automaton cannot proceed when both $A$ and $\bar{A}$ are non-empty. Assuming that this requirement is part of the semantics of the automaton (i.e., part of its transition rule) we can implement a zero test $A = 0$ by consecutively adding and subtracting from the counter $\bar{A}$, where we require that counter $\bar{A}$ is not used otherwise. Obviously, under the proposed semantics, the automaton would block on a non-zero $A$ counter while $\bar{A}$ is operated.

Motivated by this, we say that zero tests are implemented by *conflicting counters* if in the counter automaton we replace every transition $(p \to q, A = 0)$ by the transitions $(p \to q', +\bar{A})$, $(q' \to q'', \varepsilon)$ and $(q'' \to q, -\bar{A})$ using two new intermediate states $q', q''$. The middle transition is introduced for synchronization reasons motivated by the simulation by P systems described in the proofs that follow. Using this assumption, of course, we only use three types of instructions ($\varepsilon$, $+A$ and $-A$) in the counter automaton, and we only need to simulate those by the P system. On the other hand, we have to make sure that our P system

obeys the new semantics, thus it should block in the presence of both $A$ and $\bar{A}$ positive counter contents. This, however, turns out to be quite simple to achieve.

This approach is not new. We can find it in the proof of Theorem 3.3 in [Pă02b], where the rule $H_A A \to \#$ introduces a trap symbol when both $A$ and conflicting $H_A$ are present.

## 3 Single Membrane

If one allows antiport, then a single membrane suffices[1] to reach RE, improving on the previous bound of two obtained in [PP02].

**Theorem 1.**

1. $\mathbb{N}\cdot\mathsf{PP}_1(sym_1, anti_2) = \mathbb{N}\cdot\mathsf{RE}$
2. $\mathbb{N}\cdot\mathsf{PP}_1(sym_0, anti_2) = \mathbb{N}_1\cdot\mathsf{RE}$

*Proof.* We consider the inclusion $\mathbb{N}\cdot\mathsf{RE} \subseteq \mathbb{N}\cdot\mathsf{PP}_1(\mathrm{sym}_1, \mathrm{anti}_2)$, proving it by simulating a counter automaton. During computation the (single) membrane contains a state symbol, and for each counter a number of counter symbols equal to the counter value.

For the increase and decrease operations the simulation of the counter automaton can be done in a straightforward manner. Using antiport, the membrane exchanges its state for the successor state, at the same time moving in or moving out the proper counter symbol.

As for the zero test, we assume that the counter automaton is implemented using conflicting counters. Testing counter $A$ for zero, the P system brings in the symbol $\bar{A}$ representing the counter conflicting with $A$, and removes the symbol $\bar{A}$ two steps later. Together the symbols $A$ and $\bar{A}$ cause an infinite computation by bringing in the trap symbol $\#$, which is forced by the rule of maximal parallelism. Here it becomes clear why we have insisted on implementing the conflicting counters using three transitions: the delay is necessary to perform the zero test.

The system is constructed as follows:

| counter transitions: | P system rules: | |
|---|---|---|
| $(p \to q, \varepsilon)$ | $(q, \mathrm{in}; p, \mathrm{out})$ | |
| $(p \to q, +A)$ | $(qA, \mathrm{in}; p, \mathrm{out})$ | |
| $(p \to q, -A)$ | $(q, \mathrm{in}; pA, \mathrm{out})$ | |
| | $(\#, \mathrm{in}; \bar{A}A, \mathrm{out})$ | conflict |
| | $(\#, \mathrm{in}; \#, \mathrm{out})$ | trap |

At the beginning of the computation the membrane contains the single initial state symbol $p_{in}$. Note that the P system as given above reaches a halting state if the counter automaton halts in a state where it cannot decrease its counter.

[1] At the workshop in Curtea de Argeş we learned that this result was obtained independently by Freund and Oswald [FO02], using similar techniques.

This is unwanted, as the blocking (unsuccessful) computation of the counter automaton is transformed into a halting (successful) computation of the P system. To avoid this situation, we add the rule $(p, \text{in}; p, \text{out})$ for every non-final state $p$.

If we allow symport, the final state $p_f$ can be removed using the rule $(p_f, \text{out})$. According to our assumptions on the counter automaton, all its counters are empty, except for the output counter. Hence, in our simulation, the membrane only contains the symbols from the output counter.

Otherwise, for $\mathbb{N}\cdot\mathsf{PP}_1(\text{sym}_0, \text{anti}_2)$, the system halts in the final state with its state symbol present in the membrane as additional symbol. □

## 4 Symport Only

P systems with symport only are studied in [M$^+$02a], where the equalities $\mathbb{N}\cdot\mathsf{RE} = \mathbb{N}\cdot\mathsf{PP}_2(\text{sym}_5, \text{anti}_0) = \mathbb{N}\cdot\mathsf{PP}_3(\text{sym}_4, \text{anti}_0) = \mathbb{N}\cdot\mathsf{PP}_5(\text{sym}_3, \text{anti}_0)$ are obtained. This final family is improved to $\mathbb{N}\cdot\mathsf{PP}_5(\text{sym}_2, \text{anti}_0)$ in [M$^+$02b]. We obtain $\mathbb{N}\cdot\mathsf{RE} = \mathbb{N}\cdot\mathsf{PP}_2(\text{sym}_3, \text{anti}_0)$ as a new bound for computational completeness for P systems with symport. Actually, we focus here on a minimal number of membranes (under the assumption that antiport is not allowed). In the next section we try to minimize the number of symbols invlved in the symport rules.

The result $\mathbb{N}\cdot\mathsf{RE} = \mathbb{N}\cdot\mathsf{CP}_1(*, 2)$ in [Ho02, Corollary 2] states that P systems with carriers with two passengers are computationally complete. Intuitively, moving a carrier and its two passengers can be described by a symport of weight three, suggesting $\mathbb{N}\cdot\mathsf{RE} \stackrel{?}{=} \mathbb{N}\cdot\mathsf{PP}_1(\text{sym}_3, \text{anti}_0)$. Unfortunately there are some technical differences between the two types of systems that are in the way of a direct application of this result.

All in all, we obtain the following improvement[2] of previous results, slightly less satisfactory than the intuition above.

**Theorem 2.**

1. $\mathbb{N}\cdot\mathsf{PP}_2(sym_3, anti_0) = \mathbb{N}\cdot\mathsf{RE}$
2. $\mathbb{N}\cdot\mathsf{PP}_1(sym_3, anti_0) = \mathbb{N}_2\cdot\mathsf{RE}$

*Proof.* First we consider the single membrane result. The technique used to simulate a counter automaton follows the one presented for Theorem 1. Additionally, for each transition there is a transition symbol $\rho$ present in the outer membrane, and a divergence symbol $\partial$ that is used to bring the trap symbol # into the membrane.

In each simulation step the state and a matching transition move out, after which the transition takes the new state in. At the same time the appropriate counter symbol may be moved in or out as required by the transition. As before

[2] This result was obtained independently by A. Păun [Pă02a], in the 'classical way' simulating matrix grammars.

we avoid a blocking situation, by adding rules for transitions $(p \to p, \varepsilon)$ for each non-final state $p$.

| counter transitions: | P system rules: | |
|---|---|---|
| $\rho = (p \to q, \varepsilon)$ | $(\rho p, \text{out}),\ (\rho q, \text{in})$ | |
| $\rho = (p \to q, +A)$ | $(\rho p, \text{out}),\ (\rho q A, \text{in})$ | |
| $\rho = (p \to q, -A)$ | $(\rho p A, \text{out}),\ (\rho q, \text{in})$ | |
| | $(\partial \bar{A} A, \text{out})$ | conflict |
| | $(\partial \#, \text{in}),\ (\partial \#, \text{out})$ | trap |

At the end of a successful simulation all transition symbols $\rho$, final state $p_f$, and $\partial$ remain inside the membrane. Then it is possible to move all $\rho$'s out using the rules $(p_f \rho, \text{out}),\ (p_f, \text{in})$ so that finally $p_f$ and $\partial$ remain inside, adding 2 to the number generated.

If we want to obtain $\mathbb{N}\cdot\mathsf{RE}$ rather than $\mathbb{N}_2\cdot\mathsf{RE}$ we add another membrane, moving all generated output symbols inside it. □

## 5 Minimal Symports

We study systems where only symports of weight two are allowed, a minimal nontrivial case of coupled transport. In [M$^+$02a,M$^+$02b,Pă02b] the following results are obtained: $\mathbb{N}\cdot\mathsf{PP}_5(\text{sym}_2, \text{anti}_0) = \mathbb{N}\cdot\mathsf{RE}$, $\mathbb{N}\cdot\mathsf{PP}_4(\text{psym}_2, \text{anti}_0) = \mathbb{N}\cdot\mathsf{RE}$ (with *permitting* context), and $\mathbb{N}\cdot\mathsf{PP}_3(\text{fsym}_2, \text{anti}_0) = \mathbb{N}\cdot\mathsf{RE}$ (with *forbidding* context).

We generalize these results, using a construction similar to the proof of Theorem 4.1 in [M$^+$02b], again simulating counter automata rather than matrix grammars. In this construction in a central membrane the next transition to be applied is nondeterministically chosen by sending out a transition symbol. As this single symbol cannot control a synchronization of all actions needed for the simulation, it needs to be 'duplicated' in a number of intermediate membranes.

**Theorem 3.** $\mathbb{N}\cdot\mathsf{PP}_4(sym_2, anti_0) = \mathbb{N}\cdot\mathsf{RE}$

*Proof.* As before to prove the inclusion from right to left a counter automaton is simulated by a P system. The P system consists of four membranes: three nested inside each other, numbered 1 to 3 from outside in, and an additional output membrane nested inside the outer membrane 1. The outer membrane holds the symbols representing the values of the counters, but the output counter symbols are moved into the output membrane. Rules of the P system are summarized in Table 1; here we give some explanations on their function in the system.

*Initial configuration.* The environment gives an infinite supply of all counter symbols $A$, $A^-$ for each counter $A$ (including conflicting $\bar{A}$ and $\bar{A}^-$), as well as all state symbols $p$. Initially outer membrane 1 contains the initial state $p_{in}$. Membranes 2 and 3 contain symbols $\rho'$ and $\rho$ respectively, representing the each transition $\rho$. Additionally membrane 2 contains a trap symbol #, and membrane 3 contains a catalyst $c$ and a halting symbol †.

**Table 1.** The rules in the membranes according to their functions.

| | | |
|---|---|---|
| **simulation of transitions:** | | |
| in 3 | $(c\rho, \text{out}),\ (c\rho, \text{in})$ | choosing $\rho$ |
| in 2 | $(\rho\rho', \text{out}),\ (\rho\rho', \text{in})$ | duplication |
| in 1 | $(\rho p, \text{out}),\ (\rho', \text{out}),\ (\rho q, \text{in}),\ (\rho', \text{in})$ | $\rho = (p \to q, \varepsilon)$ |
| | $(\rho p, \text{out}),\ (\rho', \text{out}),\ (\rho q, \text{in}),\ (\rho' A, \text{in})$ | $\rho = (p \to q, +A)$ |
| | $(\rho p, \text{out}),\ (\rho', \text{out}),\ (\rho q, \text{in}),\ (\rho' A^-, \text{in})$ | $\rho = (p \to q, -A)$ |
| **counter behaviour:** | | |
| in 1 | $(AA^-, \text{out})$ | decrement |
| in 2 | $(A^-, \text{in}),\ (A^-\#, \text{out})$ | decrement zero |
| output | $(a, \text{in})$ | output symbol $a$ |
| **failure:** | | |
| in 2 | $(\rho, \text{in}),\ (\rho\#, \text{out})$ | consistency $\rho$-$\rho'$ |
| | $(\bar{A}A, \text{in}),\ (A\#, \text{out})$ | conflict |
| in 1 | $(\#, \text{in}),\ (\#, \text{out})$ | trap |
| **halting:** | | |
| in 3 | $(c\dagger, \text{out})$ | |
| in 2 | $(\dagger, \text{out})$ | |
| in 1 | $(p_f\dagger, \text{out})$ | halt |
| in 2 | $(p\dagger, \text{in}),\ (p\#, \text{out})$ | nonfinal state $p$ |

*Simulation of transitions.* Membrane 3 contains a transition symbol $\rho$ for each transition. Additionally, it contains the symbol $c$, that is used to carry out transition symbols, one at a time (such a symbol is elsewhere called a catalyst). This mechanism nondeterministically choses a transition to be simulated.

Membrane 2 contains a supplementary transition symbol $\rho'$ for each transition, designed to travel synchronously with $\rho$ into membrane 1. In this way two symbols duplicating the same transition in the outer membrane are obtained; one of them ($\rho$) replaces the state symbol by moving in and out, the other ($\rho'$) implements the counter instruction.

*Counter behaviour.* The values of the counters are stored in the outer membrane 1 as a number of symbols. Increasing the counter ($+A$) is implemented by bringing in a symbol from the environment. Decreasing a counter ($-A$) is *not* implemented by moving a symbol directly outside as such a rule will be ignored when the counter is zero (without blocking the corresponding change of state). Instead a 'negative' symbol $A^-$ is introduced which has the task of removing a counter symbol $A$ one step later. If $A$ is not present in the membrane, the computation blocks on empty counter, and $A^-$ moves inside membrane 2. Any counter symbol entering membrane 2 will cause an infinite (unsuccessful) computation by bringing out the trap symbol #. Zero tests are assumed to be implemented in the counter automaton by conflicting counters. Hence, we add rules to force infinite computations whenever a symbol $A$ and its conflicting counter $\bar{A}$ are both present. (Do not confuse the functions of $\bar{A}$ and $A^-$.)

*Consistency.* Once $\rho$ and $\rho'$ have arrived into membrane 1 it is expected they act together and move out to execute the transition (or to return together to membrane 2 backtracking the simulation). However, if $\rho$ does not match the

present state, it cannot go out while $\rho'$ can. If $\rho'$ moves out, maximal parallelism forces $\rho$ into membrane 2 where it releases #, causing an infinite computation.

*Halting.* The halting symbol † present in membrane 3 is used to end the computation. This symbol moves out $c$ from the membrane 3, effectively stopping the possibility of simulating other transitions. Note that $c$ can only be in membrane 3 if no transition symbol $\rho$ is outside of membrane 3 (and if no transition symbol $\rho'$ is outside of membrane 2). Finally, † moves into membrane 1 and removes the final state symbol $p_f$. The computation has ended, and membrane 1 contains only counter symbols (assuming the trap symbol was not released). By assumption, upon entering the final state the only nonempty counter is the output counter. □

In the above proof the fourth membrane was introduced for the sole purpose of having an elementary output membrane; otherwise the outer membrane could very well take the role of output membrane as it is empty at the end of the computation. Keeping the three membranes 1 to 3 as above, the only elementary membrane is 3, but that membrane contains the transition symbols $\rho$ – it needs some clever reprogramming to move these symbols out at the end of the computation. We conjecture Theorem 3 can be improved to $\mathbb{N}\mathsf{PP}_3(\mathrm{sym}_2, \mathrm{anti}_0) \stackrel{?}{=} \mathbb{N}\cdot\mathsf{RE}$.

## 6 Following the Traces

In the previous sections we have considered P systems that generate sets of numbers by counting the objects in a designated membrane at the end of a halting (successful) computation. Here we take the approach of [I$^+$02] to have P systems with objects that generate strings (over an alphabet $\Delta$) by introducing a distinguished object $t$, the *traveller*, and a labelling $h$ which assigns an element of $\Delta \cup \lambda$ to each membrane. The result of a halting computation is here the *trace* of the object $t$, which is the sequence of labels of the membranes where $t$ resides during the consecutive steps of the computation. Again, we are not very formal here, and refer to [I$^+$02], or [Pă02b, Section 4.4].

We obtain new characterizations of $\mathsf{RE}$ in terms of these trace languages of P systems as a direct application of the constructions in the previous sections. In fact, a characterization of $\mathsf{RE}$ using these systems was previously unknown, except in the case of 1-letter alphabets.

For this section we need a slight extension of our previous notions. By $\ell\cdot\mathsf{RE}$ we mean the family of recursively enumerable languages over alphabets of size $\ell$ (where for all practical matters $\mathbb{N}\cdot\mathsf{RE}$ equals $1\cdot\mathsf{RE}$). Here we must again allow the counter automaton to read letters from an input tape (and we forget about the output counter). Without loss of generality we assume that a transition either reads input, or performs a counter instruction. Hence, transitions are of the form $(p \xrightarrow{a} q, \varepsilon)$ when reading $a \in \Delta \cup \lambda$, or of the form $(p \to q, \iota)$, where $\iota$ is either $+A$ or $-A$ . As before, we omit $\iota$ equal to $A = 0$ assuming the counter automaton is implemented using conflicting counters.

The family of languages over alphabets of size $\ell$, defined by traces of P systems with symport/antiport having at most $m$ membranes, symport of weight

at most $j$, and antiport of weight at most $k$ is denoted by $\ell\cdot\mathsf{LP}_m(\mathrm{sym}_j, \mathrm{anti}_k)$. We carry the size $\ell$ of the alphabet around in the notation as the complexity of the system necessarily depends on the size of the alphabet.

We quote here some relevant results from [I+02], i.e., not involving forbidding or permitting contexts for the rules.

**Proposition 1.**

1. $3\cdot\mathsf{LP}_4(sym_2, anti_0) - \mathsf{CF} \neq \varnothing$
2. $1\cdot\mathsf{LP}_2(sym_2, anti_2) = 1\cdot\mathsf{LP}_5(sym_2, anti_0) =$
   $1\cdot\mathsf{LP}_3(sym_4, anti_0) = 1\cdot\mathsf{LP}_2(sym_5, anti_0) = 1\cdot\mathsf{RE}$

Adapting our one-letter constructions from the previous sections, we obtain the following results.

**Theorem 4.**

1. $\ell\cdot\mathsf{LP}_{\ell+1}(sym_0, anti_2) = \ell\cdot\mathsf{RE}$
2. $\ell\cdot\mathsf{LP}_{\ell+1}(sym_3, anti_0) = \ell\cdot\mathsf{RE}$
3. $\ell\cdot\mathsf{LP}_{\ell+2}(sym_2, anti_0) = \ell\cdot\mathsf{RE}$

*Proof.* Our construction is an elementary extension of the proofs of the results of previous sections. A similar observation (restricted to the one-letter case) led to the characterizations of Proposition 1(2) in [I+02, Corollary 1].

Let $\Delta$ be an alphabet with $\ell$ symbols. Consider an outer membrane containing a membrane $M_a$ for each symbol $a \in \Delta$. The label associated to $M_a$ is $a$; all other membranes get label $\lambda$. Initially the traveller $t$ resides in the outer membrane. It is taken into membrane $M_a$ by every symbol $a$ that is brought into the outer membrane as a result of the simulation of a transition reading $a$. This can be done by rules of weight two assigned to membrane $M_a$: either by symport $(at, \mathrm{in})$ and $(t, \mathrm{out})$, or by antiport $(at, \mathrm{in}; \omega, \mathrm{out})$ and $(\omega, \mathrm{in}; t, \mathrm{out})$ – assuming an additional symbol $\omega$ in each of the membranes $M_a$.

The rules $(p \to q, \iota)$ are simulated by the system as before. The rules $(p \xrightarrow{a} q, \varepsilon)$ reading input are simulated as the rule $(p \to q, +a)$ in the previous constructions. This means that a symbol for $a$ is brought into the outer membrane. This symbol then causes the traveller $t$ to visit the membrane $M_a$ the next step of the computation. The symbol $a$ stays in the membrane $M_a$ afterwards.

Note that this extension to our previous construction needs symport or antiport rules of weight two, and needs an internal membrane for each element of $\Delta$. As the system is not counting objects at the end of the computation, we may start with the optimized results for $\mathbb{N}_1\cdot\mathsf{RE}$ or $\mathbb{N}_2\cdot\mathsf{RE}$ rather than those for $\mathbb{N}\cdot\mathsf{RE}$.

Thus the result $\mathbb{N}\cdot\mathsf{PP}_1(\mathrm{sym}_0, \mathrm{anti}_2) = \mathbb{N}_1\cdot\mathsf{RE}$ from Theorem 1(2) generalizes to $\ell\cdot\mathsf{LP}_{\ell+1}(\mathrm{sym}_0, \mathrm{anti}_2) = \ell\cdot\mathsf{RE}$; the result $\mathbb{N}\cdot\mathsf{PP}_1(\mathrm{sym}_3, \mathrm{anti}_0) = \mathbb{N}_2\cdot\mathsf{RE}$ from Theorem 2(2) generalizes to $\ell\cdot\mathsf{LP}_{\ell+1}(\mathrm{sym}_3, \mathrm{anti}_0) = \ell\cdot\mathsf{RE}$.

Finally, the result $\mathbb{N}\cdot\mathsf{PP}_4(\mathrm{sym}_2, \mathrm{anti}_0) = \mathbb{N}\cdot\mathsf{RE}$ from Theorem 3 generalizes to $\ell\cdot\mathsf{LP}_{\ell+2}(\mathrm{sym}_2, \mathrm{anti}_0) = \ell\cdot\mathsf{RE}$, where we have $\ell+2$ membranes rather than the straightforward $\ell+4$ membranes by observing that the specific output membrane

is here replaced by the $M_a$ membranes, and that membrane 2 present inside the outer membrane in the previous proof may very well include the role of one of the $M_a$ membranes. In particular, for $\ell = 1$, we need only three membranes, nested inside each other to perform a simulation of the computation as in Theorem 3. No additional output membrane is needed, the middle membrane receives the terminal symbol taking in the traveller. □

Note that the string traced by the traveller can also be 'read' by looking at the consecutive symbols that are taken into the outer membrane from the environment. This is the approach taken in the P automata of [CV02]. In that paper however, there is the additional restriction that the communication of objects is one-way only, moving them deeper into the system. Quite involved constructions are needed to obtain computational completeness under that restriction.

For one letter alphabets ($\ell = 1$) we have obtained the following results that should be compared to those presented in Proposition 1.

**Corollary 1.**
$1{\cdot}\mathsf{LP}_2(sym_0, anti_2) = 1{\cdot}\mathsf{LP}_2(sym_3, anti_0) = 1{\cdot}\mathsf{LP}_3(sym_2, anti_0) = 1{\cdot}\mathsf{RE}$

In fact, the original proofs of Theorem 1 and Theorem 2 even suggest the results $1{\cdot}\mathsf{LP}_1(\mathrm{sym}_0, \mathrm{anti}_2) \stackrel{?}{=} 1{\cdot}\mathsf{RE}$ and $1{\cdot}\mathsf{LP}_1(\mathrm{sym}_3, \mathrm{anti}_0) \stackrel{?}{=} 1{\cdot}\mathsf{RE}$, as it is possible to use the outer membrane as the one labelled by $a$, rather than introducing a new one. The only reason these "results" cannot be obtained lie in the formal definitions of the P systems that we use: initially the traveller cannot be outside the membrane as the environment carries only symbols available in unbounded quantities; putting the traveller in the membrane however, makes it visible in the first step of the trace, making it impossible to generate the empty string.

## 7 Résumé and Outlook

We have illustrated how the simulation of counter automata can be profitable in obtaining universal P systems of low complexity. Their advantage over matrix grammars can be explained by the following intuition: there are less symbols involved in the counter rule $(p \to q, \pm A)$ than in the generic matrix – even in the relatively restricted grammars in binary normal form, where a typical matrix has the form $(X \to Y, A \to \alpha)$, $|\alpha| \leq 2$. It was Gheorghe Păun who realized the importance of counter automata in the context of non-rewriting P systems in [Pă00b]: "*Maybe register machines can be used in order to prove results about such purely-communicative P systems*". They now appear in several papers, for instance [CV02,FO02,So02] as presented at the WMC'02 workshop. However, for P systems that rewrite string objects it still needs to be investigated whether counter automata simulations can lead to smaller P systems.

We relate the optimal results obtained in the literature to the new bounds from the present paper.

| previous result | reference | this paper |
|---|---|---|
| $\mathbb{N}\cdot\mathsf{PP}_2(\mathrm{sym}_2,\mathrm{anti}_2)=\mathbb{N}\cdot\mathsf{RE}$ | [PP02, Thm 4.1] | $\mathbb{N}\cdot\mathsf{PP}_1(\mathrm{sym}_1,\mathrm{anti}_2)$, Thm 1 |
| $\mathbb{N}\cdot\mathsf{PP}_5(\mathrm{sym}_2,\mathrm{anti}_0)=\mathbb{N}\cdot\mathsf{RE}$ | [M$^+$02b, Thm 4.1] | $\mathbb{N}\cdot\mathsf{PP}_4(\mathrm{sym}_2,\mathrm{anti}_0)$, Thm 3 |
| $\mathbb{N}\cdot\mathsf{PP}_2(\mathrm{sym}_5,\mathrm{anti}_0)=\mathbb{N}\cdot\mathsf{RE}$ | [M$^+$02a, Thm 1] | $\mathbb{N}\cdot\mathsf{PP}_2(\mathrm{sym}_3,\mathrm{anti}_0)$, Thm 2 |
| $\mathbb{N}\cdot\mathsf{PP}_3(\mathrm{sym}_4,\mathrm{anti}_0)=\mathbb{N}\cdot\mathsf{RE}$ | [M$^+$02a, Thm 2] | $\mathbb{N}\cdot\mathsf{PP}_2(\mathrm{sym}_3,\mathrm{anti}_0)$, Thm 2 |
| $\mathbb{N}\cdot\mathsf{PP}_3(\mathrm{sym}_0,\mathrm{anti}_2)=\mathbb{N}_1\cdot\mathsf{RE}$ | [M$^+$02a, Thm 4] | $\mathbb{N}\cdot\mathsf{PP}_1(\mathrm{sym}_0,\mathrm{anti}_2)$, Thm 1 |
| $1\cdot\mathsf{LP}_2(\mathrm{sym}_2,\mathrm{anti}_2)=1\cdot\mathsf{RE}$ | [I$^+$02, Cor 1] | $1\cdot\mathsf{LP}_2(\mathrm{sym}_0,\mathrm{anti}_2)$, Cor 1 |
| $1\cdot\mathsf{LP}_5(\mathrm{sym}_2,\mathrm{anti}_0)=1\cdot\mathsf{RE}$ | [I$^+$02, Cor 1] | $1\cdot\mathsf{LP}_3(\mathrm{sym}_2,\mathrm{anti}_0)$, Cor 1 |
| $1\cdot\mathsf{LP}_3(\mathrm{sym}_4,\mathrm{anti}_0)=1\cdot\mathsf{RE}$ | [I$^+$02, Cor 1] | $1\cdot\mathsf{LP}_2(\mathrm{sym}_3,\mathrm{anti}_0)$, Cor 1 |
| $1\cdot\mathsf{LP}_2(\mathrm{sym}_5,\mathrm{anti}_0)=1\cdot\mathsf{RE}$ | [I$^+$02, Cor 1] | $1\cdot\mathsf{LP}_2(\mathrm{sym}_3,\mathrm{anti}_0)$, Cor 1 |

We conjecture that Theorem 3 can be improved to $\mathbb{N}\cdot\mathsf{PP}_3(\mathrm{sym}_2,\mathrm{anti}_0)\stackrel{?}{=}\mathbb{N}\cdot\mathsf{RE}$. Of course, it would be interesting to further investigate the optimality of the results, and to discover strict (but nontrivial) subfamilies of RE. Most interesting are the families $\mathbb{N}\cdot\mathsf{PP}_2(\mathrm{sym}_2,\mathrm{anti}_0)$ and $\mathbb{N}\cdot\mathsf{PP}_2(\mathrm{sym}_2,\mathrm{anti}_1)$, not covered by these investigations.

Hence we only partially answer to the open problem Q8 present in [Pă02b], having improved the old results, but not considering optimality of the present characterizations. Problem Q9 concerning the power of systems with only one membrane has been given a unexpected answer. The related question, on infinite sets in $\mathbb{N}\cdot\mathsf{PP}_1(\mathrm{sym}_2,\mathrm{anti}_1)$, remains open. Problem Q12 has been answered by our Theorem 4, giving a characterization of RE languages using trace languages, even without permitting or forbidding context. Finally, the P systems with symport/antiport can be interpreted as *networks of membranes*, the subject of Chapter 6 in [Pă02b]. There, our Theorem 3 leads to (minor) improvements to Corollaries 6.2.1 and 6.2.2 in [Pă02b]. More importantly, our Theorem 1 implies that $\mathbb{N}\cdot\mathsf{OtP}_{1,1}(\mathrm{sym}_1,\mathrm{anti}_2)$ equals $\mathbb{N}\cdot\mathsf{RE}$ contradicting the conjecture that $\mathbb{N}\cdot\mathsf{OtP}_{*,1}(\mathrm{sym}_*,\mathrm{anti}_*)$ contains only semi-linear sets.

Note that our methods are not suited to improve on results that determine the P system complexity of other Chomsky families, like Proposition 1(1), as the counter data structure simulated here is particular to RE. Of course, one-counter languages form a proper subclass of the context-free languages, but the number of counters has not been an issue for the complexity parameters considered (number of membranes and weight of rules).

Our research did not deal with further control of the application of rules, like permitting or forbidding context. Observe that in our constructions the transition symbols $\rho$ are used in a permitting role. Having them as explicit side condition may further reduce the complexity of the rules.

*Definitional Remarks.* It has become clear in these investigations that small details in the definition of P systems with symport have their effect on the parameters in our results. We summarize them here to facilitate a discussion on these matters.

1. Symport is not considered a degenerated case of antiport: the rule $(x,\mathrm{in})$ is not equated with $(x,\mathrm{in};\lambda,\mathrm{out})$, cf. Theorem 1, where a final symbol has to be

removed from the membrane. It is impossible to do so with an antiport rule (if output 0 is being generated) and the single rule introduced for this action shows up in the result. See also Theorem 4(1) where we had to introduce a new symbol $\omega$ to artificially turn symport into antiport.
2. It would be a natural alternative to define the complexity of an antiport rule $(x, \text{in}; y, \text{out})$ as $|x| + |y|$, the total number of objects moved, rather than $\max\{|x|, |y|\}$ as is now customary. This alternative is closer to the original definitions in [PP02].
3. In determining the output, all symbols matter, not only a designated output symbol, cf. Theorem 2, where two additional symbols remain inside the membrane. We now have two variants of the result. Either we are content with the fact that only numbers at least two can be generated, or we insist on the full set $\mathbb{N}$ at the cost of an extra membrane. Alternatively we could only count occurrences of a specified output symbol. This would yield $\mathbb{N}$ with a single membrane.
4. The membrane where output is collected, is assumed to be elementary, cf. Theorem 3, where this assumption leads to the introduction of the fourth membrane for this reason only.
5. As noted after Theorem 4 and its Corollary, the traveller is always visible in the first step of a one-membrane P system. Allowing the traveller to be present in a single copy in the environment would optimize this result.

**Acknowledgements**

We are indebted to Gheorghe P for encouragement, and to the referee for his constructive comments.

## References

BibP. Bibliography of P Systems, at: *The P Systems Web Page*, C. Zandron (Ed.), http://psystems.disco.unimib.it

CV02. E. Csuhaj-Varjú, G. Vaszil. P Automata, in [PZ02], pages 177-192.

FO02. R. Freund, M. Oswald. P Systems with Activated/Prohibited Membrane Channels, *this volume*.

FP01. R. Freund, Gh. Păun. On the Number of Non-terminals in Graph-controlled, Programmed, and Matrix Grammars, in [MR01], pages 214–225.

Ho02. H.J. Hoogeboom. Carriers and Counters; P systems with Carriers vs. (Blind) Counter Automata, in: M. Ito, M. Toyama (eds.) Developments in Language Theory 2002, Preproceedings, 2002, pages 255-268. Updated paper to appear in LNCS.

HU79. J.E. Hopcroft, J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

I$^+$02. M. Ionescu, C. Martín-Vide, A. Păun, Gh. Păun. Membrane Systems with Symport/Antiport: (Unexpected) Universality Results, in: M. Hagiya, A. Ohuchi (eds.) *Preliminary Proceedings 8th International Meeting on DNA Based Computers*, 2002, pages 151–160.

MR01. M. Margenstern, Y. Rogozhin (eds.) *Procedings Universal Computational Models*, Chişinău, *Lecture Notes in Computer Science*, Volume 2055, Springer Verlag, 2001.

MP01. C. Martín-Vide, Gh. Păun. Computing with Membranes (P Systems): Universality Results, in [MR01], pages 214–225.

M$^+$02a. C. Martín-Vide, A. Păun, Gh. Păun. On the Power of P Systems with Symport Rules. *Journal of Universal Computer Science* 8 (2002), pages 317–331.

M$^+$02b. C. Martín-Vide, A. Păun, Gh. Păun, G. Rozenberg. Membrane Systems with Coupled Transport: Universality and Normal Forms. *Fundamenta Informaticae* 49 (2002), pages 1-15.

Mi61. M.L. Minsky. Recursive Unsolvability of Post's Problem of "Tag" and Other Topics in Theory of Turing Machines. *Annals of Mathematics*, 74 (1961), pages 437–455.

Păun02a. A. Păun. Membrane Systems with Symport/Antiport. Universality Results, in [PZ02], pages 333–343.

PP02. A. Păun, Gh. Păun. The Power of Communication: P Systems with Symport/Antiport. *New Generation Computing*, 20 (2002), pages 295–305.

Păun00a. Gh. Păun. Computing with Membranes. *Journal of Computer and System Sciences*, 61 (2000), pages 108–143.

Păun00b. Gh. Păun. Computing with Membranes (P Systems): Twenty Six Research Topics, CDMTCS TR 119, Univ. of Auckland, 2000. http://www.cs.auckland.ac.nz/CDMTCS/

Păun02b. Gh. Păun. *Membrane Computing. An Introduction*. Natural Computing Series, Springer Verlag, 2002.

PR02. Gh. Păun, G. Rozenberg. A Guide to Membrane Computing. *Theoretical Computer Science*, 287 (2002), pages 73–100.

PZ02. Gh Păun, C. Zandron (eds.) *Pre-proceedings of the Second Workshop on Membrane Computing*, Curtea de Argeş Romania, Molconet Publication 1, 2002. Updated papers to appear in LNCS, *this volume*.

So02. P. Sosík, P Systems Versus Register Machines: Two Universality Proofs, in [PZ02], pages 371-382.

# Towards a Hierarchy of Conformons–P Systems*

Pierluigi Frisco[1] and Sungchul Ji[2]

[1] L.I.A.C.S., Leiden University
Niels Bohwerg 1, 2333 CA Leiden, The Netherlands
pier@liacs.nl
[2] Dep. of Pharm. and Toxic
Rutgers University, Piscataway, N.J. 08855 U.S.A.
sji@eohsi.rutgers.edu

**Abstract.** The combination of a theoretical model of the living cell and membrane computing suggested a new variant of a computational model called conformons-P system based on membrane-enclosed compartments where conformon-like objects evolve. Some variants of such model are presented and their computational power is sketched.

## 1 Introduction

One of the direction of research of formal language theory in the last years saw the creation of new computability models directly inspired by biochemical processes. One of the first papers in this direction [6] introduced and investigated the computational power of *splicing* (a recombinational behavior of DNA and RNA molecules allowed by specified classes of enzymatic activities) from a mathematical point of view. Since this result other similar models have been introduced having variants in the structure of the systems itself, in the kind of performed operations, and so on.

These investigations expound the theoretical facet of *biomolecular computing*.

Our research resides in this sphere. Inspired by some basic principles of *biocybernetics* [10] (a general molecular theory of living processes), we introduced [4] a new model of *membrane systems* that we study here in more detail.

In [10] biocybernetics is formulated on the basis of principles, concepts and analogies imported from physics, chemistry and cybernetics. The most novel physical concept to emerge in that theory is that of *gnergy*, a hybrid physical entity composed of free energy and genetic information that is postulated to be ultimately responsible for driving all molecular machines. Discrete physical entities carrying gnergy are called *gnergons* and there are two examples of gnergons that have been identified in biology so far: *conformons*, sequence-specific mechanical strains of biopolymers, and *IDSs* (intracellular dissipative structures) intracellular chemical and mechanical stress gradients and waves. Conformons and IDSs are utilized to formulate what appears to be the first coherent theoretical model of the living cell known as the *Bhopalator* [9,13].

* Work partially supported by contribution of EU commission under The Fifth Framework Programme, project "MolCoNet" IST-2001-32008

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 302–318, 2003.

Conformons (explained in Section 2) are visualized as a collection of a small number of catalytic residues of enzymes or segments of nucleic acids that are arranged in space and time with appropriate force vectors so as to cause chemical transformations or physical changes on a substrate (during catalysis) or a bound ligand (during the control of gene expression).

*Membrane systems* (also called *P systems*) are a new class of distributed and parallel computing devices introduced in [21]. In the seminal paper the author considers systems based on a hierarchically arranged, finite *cell-structure* consisting of several cell-membranes embedded in a main membrane called the *skin*. The membranes delimit *regions* where *objects*, elements of a finite set, and evolution rules can be placed.

The objects evolve according to given *evolution rules* associated with a region, and they may also move between regions. A *computation* starts from an initial configuration of the system, defined by a cell-structure with objects and evolution rules in each cell, and terminates (*halts*) when no further rule can be applied.

It is possible to assign a result to a computation in two ways: (1) a multiset, considering the multiplicity of objects present in a specific (*output*) membrane in a halting configuration, or (2) a set of strings, composed of the strings over a specific alphabet sent out of the system. Combining the outputs of each possible computation the behaviour of the system is obtained, a multiset-language (a set of numbers) or a string-language (a set of strings).

In [21] the author examines three ways to view P systems: transition, rewriting and splicing P systems. Starting from these, several variants were considered (see for instance [2,16,17,22,24]). Each of these variants has been shown to generate recursively enumerable sets or vectors of natural numbers. The latest information about P systems can be found at the url `http://psystems.disco.unimib.it/`.

*Conformons-P systems*, the variant of P systems presented in Section 3, consider as objects conformons - an ordered pair of name and value. This way to consider objects differs in a substantial way from the others described in the literature of membrane systems. Until now an object has been considered either as simple entity without internal structure or a string, that is an entity with a well defined structure. Objects considered in our research may be placed in between these two categories as the only structure related to the name is its value.

The generative power of the basic model of conformons-P systems is investigated in Section 4, while Sections 5 and 6 describe the generative power of variants of it.

In Section 7 we outline some other possible variants of conformons-P systems. Moreover in that section we delineate how this model may be used to describe and study the massive parallelism so fundamental in biomolecular computing and to which fields the computation based on conformons might be extended.

We omit here the proofs of our results. The interested reader may refer to [3].

## 2 Conformons in Molecular Biology

Just as biology is continuing to serve as a rich source of ideas and inspirations for computer scientist interested in developping novel computational models, computer science may play an essencial role in solving fundamental problems in molecular cell biology in $21^{st}$ century [14], and as inspirations for computer scientists aspiring to discover and design novel computational frameworks but may also absolutely require computer science to formalize and test its theories in order to solve the mysteries of life on the molecular and cellular levels. The present contribution is primarily concerned with the former aspect of the computer science-biology interactions, and the latter aspect has been dealt with elsewhere by one of us [9,10,11,13].

One of the basic concepts to develop in molecular biology during the past three decades is the notion of *conformons*, defined as sequence-specific mechanical strains embedded in biopolymers, such as DNA supercoils and protein conformational deformations, that provide both the free energy and information needed for biopolymers to drive molecular processes essential for life [8,12]. The free energy content of conformons has been estimated to be in the range of 5 $\sim$10 Kcal/mole in proteins and 500 $\sim$ 2,000 Kcal/mole in DNA, while the information content per conformon has been calculated to be in the range of 20 $\sim$ 40 bits (note that 20 $\sim$ 200 bits as reported in [11] is an error) in proteins and 200 $\sim$ 600 bits in DNA [10,12].

Conformons and conformon-like entities invoked in the biological literature during the past three decades have been classified into 10 families based on their biological functions, including the origination of life, thermal fluctuations, substrate and product bindings, formation of the transition-state complex, free energy transfer, DNA replication, timing in proteins, and timing in DNA [12]. Given such a multiplicity of conformon families, each with a large number (from $10^3$ to $10^6$ ?) of members, it is possible, at least in principle, to account for all living processes in the cell in molecular terms.

This has led to the postulates (1) that the number of conformons active in and utilized by living cells are finite in number and (2) that conformons are quanta of biological actions, akin to quanta of action in quantum mechanics [12].

Another fundamental feature of the living cell, postulated to be the smallest molecular computing system in nature [11], is the biological membranes consisting of a phospholipid bilayer of about 50 angstroms (i.e., $50 \times 10^{-8}$ cm) in thickness with many different kinds of proteins, either attached to its surface or deeply embedded in it.

The basic function of biomembranes is to divide the Euclidean space into multiple compartments, to be referred to as *membrane-enclosed compartments* or simply as *membranes* when there is no danger of ambiguity. The principle of biological membranes began to be capitalized in developing new computing paradigms during the past several years, leading to the development of membrane computing [21,22,23,24].

## 3 Basic Definitions

As indicated in the Introduction, the basic ideas underlying conformons and the interactions between them have inspired us to define a new computability model. What in biocybernetics is a pair of information and free energy in this section is defined from a mathematical point of view as an ordered pair name-value, the interaction between two conformons is modeled as passage of the whole or a part of the value from one pair to another one.

Let V be a finite alphabet and $\mathbf{N}$ the set of natural numbers. A *conformon* is an element of the relation *name-value*: $V \times \mathbf{N}_0$ (where $\mathbf{N}_0 = \mathbf{N} \cup \{0\}$), denoted by $[X, x]$. We will refer to $x$ as the *value* of $[X, x]$ and to $X$ as the *name* of the conformon $[X, x]$. The symbol $X$ will also refer to the conformon itself; the context will help the reader to understand when we refer only to the name aspect of the conformon or to the whole conformon. Moreover let $r = \langle A, e, B \rangle$, $A, B \in V$, $e \in \mathbf{N}$, be a *rule* (also indicated as $A \xrightarrow{e} B$) defining the passage of (part of the) value from one conformon to another so that:

$$\begin{matrix} [A, a] & & [A, a-e] \\ & \Rightarrow_r & \\ [B, b] & & [B, b+e] \end{matrix} \tag{1}$$

with $a, b \in \mathbf{N}_0$, $a \geq e$ indicating that $[A, a]$ and $[B, b]$ *interact* according to $r$. Informally this means that $e$ is subtracted from the value of the conformon (with name) $A$ and $e$ is added to the value of the conformon (with name) $B$ only if the value of $A$ is at least $e$.

A *multiset* (over $V$) is a function $M : V \to \mathbf{N}_0$; for $d \in V$, $M(d)$ defines the *multiplicity* of $d$ in the multiset M. We will indicate this also with $(d, M(d))$. In case the multiplicity of an element of a multiset is 1 we will indicate just the element. The *support* of a multiset M is the set $supp(M) = \{d \in V \mid M(d) > 0\}$. Informally we will say that an element belongs to a multiset $M$ if it belongs to the support of $M$.

Let $M_1, M_2 : V \to \mathbf{N}_0$ be two multisets. The *union* of $M_1$ and $M_2$ is the multiset $M_1 \cup M_2 : V \to \mathbf{N}_0$ defined by $(M_1 \cup M_2)(d) = M_1(d) + M_2(d)$, for all $d \in V$. The *difference* $M_1 \backslash M_2$ is here defined only when $M_2$ is included in $M_1$ (which means that $M_1(d) \geq M_2(d)$ for all $d \in V$) and it is the multiset $M_1 \backslash M_2 : V \to \mathbf{N}_0$ given by $(M_1 \backslash M_2)(d) = M_1(d) - M_2(d)$ for all $d \in V$.

A *basic conformons-P system* of *degree* $m, m \geq 1$, is a construct $\Pi = (V, \mu, fin, ack, L_1, \ldots, L_m, R_1, \ldots, R_m)$, where $V$ is an alphabet; $\mu = (N, E)$ is a *direct labeled graph* underlying $\Pi$. The set $N \subset \mathbf{N}$ contains *vertices*, for simplicity we define $N = \{1, \ldots, m\}$. Each vertex in $N$ defines a *membrane* of the system $\Pi$. The set $E \subseteq N \times N \times pred(\mathbf{N}_0)$ defines directed labeled *edges* between vertices, indicated by $(i, j, pred(n))$ where for each $n \in \mathbf{N}$ we consider $pred(n) = \{\geq n, \leq n, = n\}$ set of *predicates*. For $x \in \mathbf{N}_0$, $p \in pred(n)$, $p(x)$ may be $(\geq n)(x)$ or $(\leq n)(x)$ or $(= n)(x)$ (only one of them), indicating $x \geq n$, $x \leq n$ and $x = n$ respectively. The membrane $fin \in N$ defines the *final membrane* while $ack \in N$ the *acknowledgment membrane* that is initially empty.

The multisets $L_i$ over $V \times \mathbf{N}_0$, $i \in N$, contain conformons; $R_i$, $i \in N$, are finite sets of rules.

Two conformons present in a membrane $i$ may interact according to a rule $r$ present in the same membrane such that the multiset of conformons $M_i$ changes into $M_i'$. So, for $i \in N$, $[A, a], [B, b] \in M_i$ and $r = \langle A, e, B \rangle \in R_i$, $A, B \in V$, $a, b, e \in \mathbf{N}_0$, we have what indicated in (1) so that $M_i' = (M_i \backslash \{[A, a], [B, b]\}) \cup \{[A, a - e], [B, b + e]\}$.

A conformon $[X, x]$ present in a membrane $i$ may *pass* to a membrane $j$ if $(i, j, p) \in E$ and $p(x)$ holds, changing the multisets of conformons $M_i$ and $M_j$ to $M_i'$ and $M_j'$ respectively. In this case $M_i' = M_i \backslash \{[X, x]\}$ and $M_j' = M_j \cup \{[X, x]\}$. The fact that the passage of an object to a membrane is regulated by some features present in the compartments themselves is already discussed by others in literature when P systems with electrical charge and variable thickness have been considered [23] or only communication was used to compute [20].

The application of a rule and the passage of a conformon from one membrane to another are the only *operations* that may be performed by a conformons-P system. A conformon present in a membrane may be involved in one of these two operations or none of them.

It is important to notice that at this stage of our investigation there is no priority between the passage of a conformon to another membrane and the application of a rule.

If a conformon may pass to another membrane or interact with another conformon according to a rule, then one of the two operations or none of them is nondeterministically chosen. So the feature "all the objects which can evolve should evolve", present in most of the other variants of P system introduced until now, is not applied here. The presence of such a universal clock, common in digital computers but not in biological processes, is very powerful from a computational point of view as it forces the system to a maximal parallelism. The parallelism of conformons-P systems is not limited by this choice of us as it is always possible that an operation is performed when it can be performed.

The possibility to carry out one of the two allowed operations in a same membrane or none of them let conformons-P systems to be nondeterministic. Nondeterminism may also arise from the configurations of a conformons-P system if in a membrane a conformon may interact with more than one conformon.

A *configuration* of $\Pi$ is an $m$-tuple $(M_1, \ldots, M_m)$ of multisets over $V \times \mathbf{N}_0$. The $m$-tuple $(L_1, \ldots, L_m)$, $supp(L_{ack}) = \emptyset$, is called *initial configuration* while any configuration having $supp(M_{ack}) \neq \emptyset$ is called *final configuration*. For two configurations $(M_1, \ldots, M_m), (M_1', \ldots, M_m')$ of $\Pi$ we write $(M_1, \ldots, M_m) \Rightarrow (M_1', \ldots, M_m')$ indicating a *transition* from $(M_1, \ldots, M_m)$ to $(M_1', \ldots, M_m')$ that is the parallel application of operations or of none of the operation in each membrane of $\mu$. If no operation is applied to a multiset $M_i$, then $M_i = M_i'$. The reflexive and transitive closure of $\Rightarrow$ is indicated by $\Rightarrow^*$.

A *computation* is a finite sequence of transitions between configurations of a system $\Pi$ starting from $(L_1, \ldots, L_m)$. The result of a computation is given by the multisets of conformons present in membrane $fin$ when any conformon

is present in membrane *ack*. When this happens the computation is halted, that is no other operation is performed even if it could. This feature is new in the area of membrane computing: it provides an alternative to the way of defining successful computations as halting computations. When a conformon is present in the acknowledge membrane the multisets of all conformons present in membrane $fin$ define the *number generated* by $\Pi$. The set of all such numbers is denoted by $L(\Pi)$.

More formally,

$$L(\Pi) = \{supp(M_{fin}) \mid (L_1, \cdots, L_m) \Rightarrow^* (M'_1, \cdots, M'_m) \Rightarrow^* (M_1, \cdots, M_m), \\ supp(M'_{ack}) = \emptyset, supp(M_{ack}) \neq \emptyset\}.$$

In this writing it is understood that $(M_1, \cdots, M_m)$ is the first configuration in the computation where the acknowledging membrane is not empty.

In the rest of this paper we analyze the computational power of basic conformons-P systems and some variants of them.

## 4 The Power of Basic Conformons-P Systems

A *module* is a group of membranes in a conformons-P system able to perform a specific task. In the figures representing conformons-P systems in this paper, modules are depicted as unique vertices with a thicker line. Such modules will have a *label* indicating the kind of module. A subscript is added to differentiate labels referring to the same kind of module present in one system. Membranes with a dashed line present in the figures representing modules indicates membranes placed outside the module.

The following result, already present in [4], is here recalled for completeness.

**Lemma 1.** *(Splitter) There exists 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 present in a specific membrane of it, may pass such a conformon to other specific membranes according to its value* $x$.

Figure 1.(a) illustrates a detailed splitter (notice that its underlying structure is a binary tree), Figure 1.(b) depicts the module representation of a Splitter having **spl** as label.

In the figures present in this paper rules are written inside the membrane they belong to. The initial configuration is represented by conformons written in bold. The rest of the conformons present in each membrane are the ones which may be present during the computation. A slash (/) separates the different value contents that a conformon may have during a computation, or the different predicates of edges connecting two membranes (in this case just one edge is depicted). When $[A, a]$ *or* $[B, b]$ is present in a membrane it indicates that one of the two conformons may be present in the membrane, but not together.

**Lemma 2.** *(Separator) There exists a module that when conformons of the type* $[X_i, x]$, $i \in \{1, \ldots, h\}$, $x \geq 1$, *are present in a specific membrane of it, may pass them to specific different membranes according to their name content.*

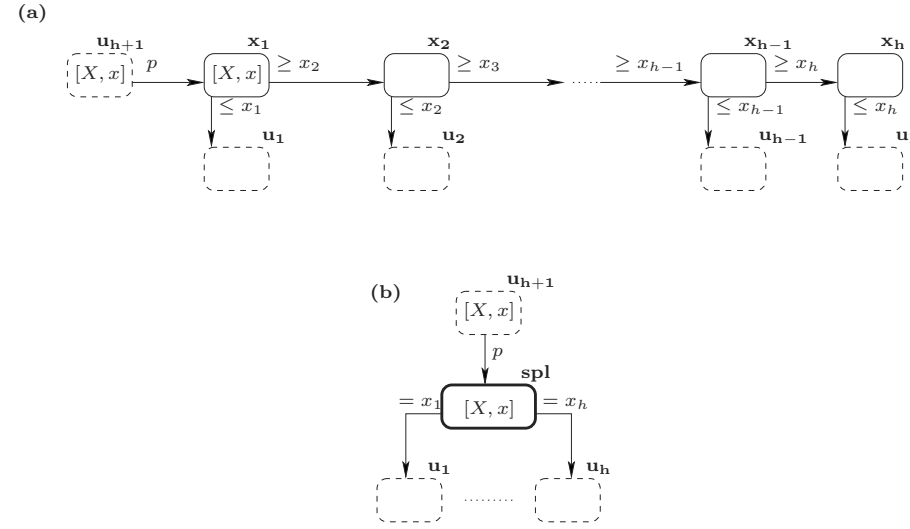



**Fig. 1.** A Splitter

*Sketch of the proof.* The number of membranes and the conformons present in this module depend on $h$. In order to differentiate the conformons $[X_i, x]$, $i$ is added to the value of them. Then the separation process is performed via a Splitter and after it their value content is reduced back to $x$. □

In Figure 2.(a) a detailed Separator is depicted, while 2.(b) illustrates its module representation having **[x, s]sep** as label. The operations performed by a Separator may be indicated in a much convenient way with a label on an edge. Figure 2.(c) represents a conformons-P system having edges with labels $[X_i, x]$ indicating that a conformon $[X_j, x]$ may pass from membrane $u$ to $u_i$ only if $j = i$.

**Lemma 3.** *(Decreaser/Increaser) There exists a module that when a conformon $[X, x]$ with $x \geq 1$ is present in a specific membrane of it may decrease or increase the value of such conformon to $q$, so that $[X, q]$ may pass to another specific membrane.*

*Sketch of the proof.* The value content of $[X, x]$ is nondeterministically decreased or increased one unit per time. When it reaches the value $q$ the conformon $[X, q]$ may pass via a Separator to a specific membrane. □

In Figure 3.(a) a detailed Decreaser/Increaser is depicted, while Figure 3.(b) illustrates its module representation having **[q]dec** as label if $x > q$ and **[q]inc** if $x < q$.

**Theorem 1.** *A basic conformons-P system may generate any set $\{0, \ldots, k\} \subset \mathbf{N}_0$.*

*Sketch of the proof.* Figure 4 represents the basic conformons-P system related to this description of the proof. Let us imagine that $k$ occurrences of a conformon

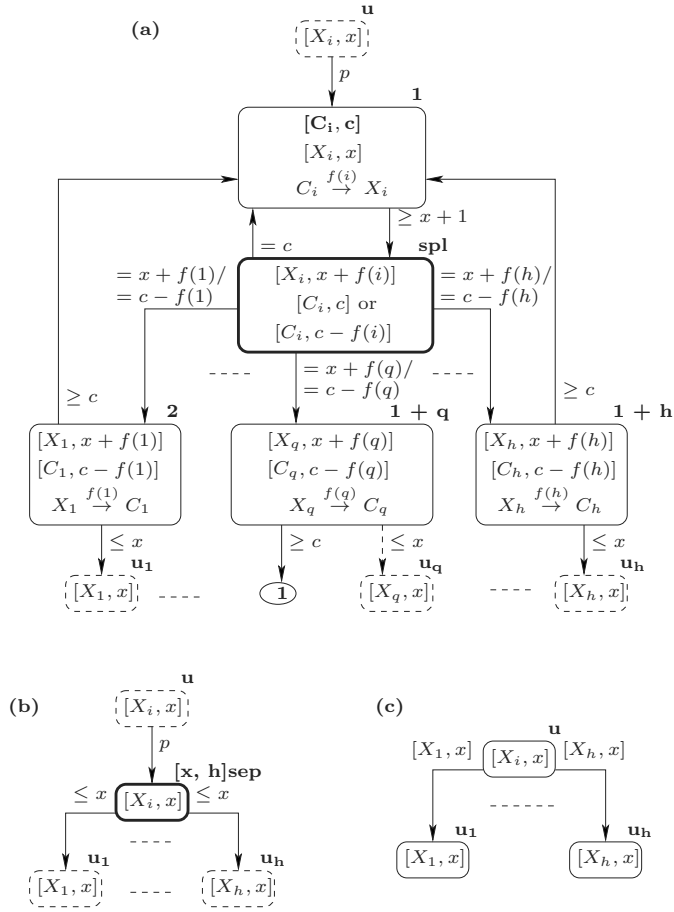



**Fig. 2.** A Separator

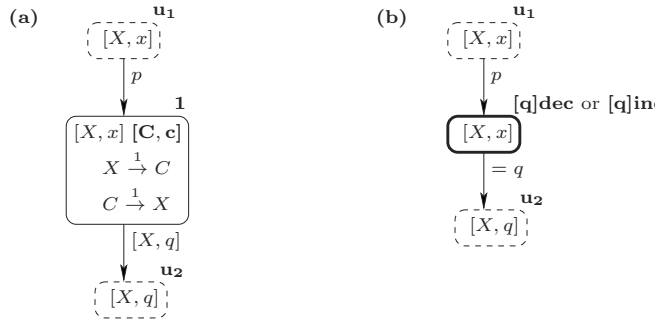



**Fig. 3.** A Decreaser

$C$ are present in membrane 1. A conformon $[X, 3]$, also present in membrane 1, may repeatedly interact with one occurrence per time of $C$, so to permit its passage to membrane 2. At any moment the conformon $X$ may pass to membrane 3, the acknowledge membrane. The result of a computation is given by the conformons $[C, 0]$ present in membrane 2, the final membrane of the system. □

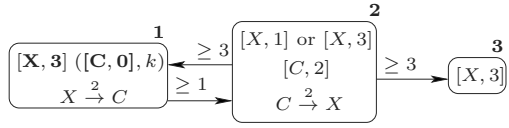



**Fig. 4.** The basic conformons-P system related to Theorem 1

It is interesting to note that the underlying graph of the basic conformons-P system present in Figure 4 is a tree and that only one kind of predicate ($\geq n,\ n \in \{1, 3\}$) is used.

## 5 Unbounded Number of Conformons

It is also possible to consider a multiset as a function $M : V \to \mathbf{N}_0 \cup \{+\infty\}$ changing consequently the definition of union and difference of two multisets. At this stage of our research we limit to 0 the value of conformons present in infinite copies. This different definition has important consequences on the generative capability of conformons-P systems (the prefix *basic* is removed when unbounded occurrences of conformons are considered).

As direct consequence of Theorem 1, we obtain:

**Corollary 1.** *A conformons-P system may generate any set* $\{0, \ldots, +\infty\} \subseteq \mathbf{N}_0$.

This corollary does not describe the maximal generative power of conformons-P systems. We can achieve this if we consider partially blind program machines.

Non-rewriting Turing machines were introduced by M. L. Minsky in [18] and then reconsidered in [19] under the name of *program machines*. After their introduction such machines and some variants of them have been studied under different names: in [5] they were called *(multi)counter machines*, in [1] *multipushdown machines*, in [15] *register machines* and in [7] *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 may 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.

Formally a *program machine* with $n$ counters ($n \in \mathbf{N}$) 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* states, $R \subseteq (s, op(l), v, w),\ s, v, w \in S, s \neq f,\ op(l) \in \{l_+, l_-\},\ 1 \leq l \leq n$, is set of *instructions* of the following form:

- $(s, l_-, v, w)$: in state $s$ if the content of counter $l$ is greater than 0, then subtract 1 from that counter and change state into $v$, otherwise change state into $w$;
- $(s, l_+, v, v)$: in state $s$ add 1 to counter $l$ and change state into $v$.

A *configuration* of a program machine $M$ with $n$ counters is given by an $n+1$-tuple from $S \times \mathbf{N}_0^n$. Given two configurations $(s, l_1, \ldots, l_n)$, $(r', l_1', \ldots, l_n')$ we define a *computational step* as $(s, l_1, \ldots, l_n) \vdash (s', l_1', \ldots, l_n')$ if $(s, op(l), v, w) \in R$ and:

- if $op(l) = l_-$ and $l_i \neq 0$, then $s' = v,\ l_i' = l_i - 1,\ l_j' = l_j,\ j \neq i,\ 1 \leq j \leq n$,
  if $op(l) = l_-$ and $l_i = 0$, then $s' = w,\ l_j' = l_j,\ 1 \leq j \leq n$.
- if $op(l) = l_+$, then $s' = v, l_i' = l_i + 1,\ l_j' = l_j,\ j \neq i,\ 1 \leq j \leq n$.

The reflexive and transitive closure of $\vdash$ is indicated by $\vdash^*$.

A *computation* is a finite sequence of transitions between configurations of a program machine $M$ starting from the initial configuration $(s_0, l_1, \ldots, l_n)$ with $l_1 \neq 0,\ l_j = 0,\ 2 \leq j \leq n$. If the last of such configurations has $f$ as state, then we say that $M$ *accepted* the number $l_1$. The set of numbers accepted by $M$ is defined as $L(M) = \{l_i \mid (s_0, l_1, \cdots, l_n) \vdash^* (s', l_1', \cdots, l_n') \vdash^* (f, l_1'', \cdots, l_n''), s' \neq f\}$. For every program machine it is possible to create another one accepting the same set of numbers and having all counters empty in the final state.

*Partially blind program machines* were introduced in [5] and defined as program machines without test on zero. The only allowed operations are increase and decrease the counters by 1 at a time. In case the machine tries to subtract from a counter having value zero it stops in a non final state. In [5] it is also proved that such machines are strictly less powerful than non blind ones.

**Theorem 2.** *The class of numbers generated by conformons-P systems coincides with the one generated by partially blind program machines.*

*Sketch of the proof.* A conformons-P system may simulate any partially blind program machine. Figure 6 represents a conformons-P system simulating a partially blind program machine, during this description of the proof we will refer to such figure.

In the initial configuration of the conformons-P system for each counter $l$ of the program machine there are unbounded occurrences of a conformon $[l, 0]$ present in membrane 3 while the final membrane (4 in the figure) contains as many copies of such conformons as the value $k_l$ of the counters at the initial configuration of the simulated machine.

Considering that the passage of conformons between membranes is determined only by the value present in a conformon, the value of specific conformons may be increased so that they may move to different membranes through

Splitters in order to perform different tasks. Only one of such conformons per time, related to the simulated instruction of the simulated machine, may move through the system so to perform the operation associated with it. The state $s_i$ of the partially blind program machine that is simulated is represented by a conformons $[s_i, 7]$ present in membrane 1. According to the instruction that is simulated the value of such a conformon may be decreased by 4, 5 of 6. In the first case a conformomn $[s'_j, 4]$ (related to the final state of the simulated machine) is created. It may pass to the acknowledgement membrane (membrane 2 in the figure) ending in this way the computation.

The simulation of adding a unit to a counter $l$ is simulated passing an occurrence of a conformon $l$ from membrane 3 to membrane 4, the opposite direction if a unit is subtracted. These two simulations are performed by $[s'_{j,l}, 6]$ or $[s'_{j,l}, 5]$ respectively. The simulation of subtraction of one unit from a counter $l$ leads to a deadlock in case the counter is empty (no occurrence of $[l, 0]$ is present in membrane 4).

A partially blind program machine may simulate any conformons-P system. We know that the number of membranes, the different conformons and the total amount of values are finite quantities in any conformons-P system. Each counter of the program machine will be labeled $O_{X_x}$ where $O$ indicates a membrane, $x$ the value content and $X$ the name content of the conformons present in the simulated system. There will be no counters representing conformons present in unbounded copis (with 0 as value) present in a membrane. The number recorded in a counter $O_{X_x}$ indicates how many copies of the conformon $[X, x]$ are present in membrane $O$.

In the initial configuration of the partialy blind program machine the counter $l_1$ contains a random number $y$. We will say that the program machine performed a correct simultion if when in a final state all counters will be empty.

The simulation is composed by three phases.

In the first phase the initial configuration of the simulated system is created. This is performed by instruction adding to each counter $O_{X_x}$ the value corresponding to the number of occurrences of the conformon $[X, x]$ present in membrane $O$ in the initial configuration.

In the second phase the simulation of possible operations of the simulated conformons-P system is performed. The passage of a conformon $[X, x]$ from membrane $U$ to membrane $Q$ is represented removing one unit from the counter $U_{X_x}$ and adding one unit to the counter $Q_{X_x}$. The interaction between two conformons is represented in a similar way. The several operations possible in the simulated system are simulated in a nondeterministic way. In case the partially blind program machine tries to simulate an operation that is not allowed in the simulated system it stops in a non final state. When the partially blind program machine adds one unit to one of the counters $O_{X_x}$, where $O$ is the label of the acknowledge membrane of the simulated system, the third phase starts.

In the third phase all registers related to conformons of the simulates conformons-P system present in membranes different than the final one are randomly decreased. The ones related to conformons present in the final membrane are

decreased together with the content of register $l_1$: for one unit removed to one of these registers one unit is subtracted from $l_1$. If in this process the machine tries to subtract one unit from a register whose content is 0 then it stops in a non final state. At any stage of this process the machine may go into the final state $f$. If then all counters are empty we will say that the partially blind program machine correctly simulated the conformons-P system accepting the number $y$. □

In case we assume that the passage of conformons to another membrane can be performed only if no interaction is possible in a membrane, then conformons-P systems (enriched with the suffix *with priorities*) may simulate any program machine. This result, already presented in [4], is here recalled for completeness.

**Theorem 3.** *The class of numbers generated by conformons-P systems with priorities coincides with the one generated by program machines.*

It is interesting to note that the just described conformons-P system and conformons-P system with priorities increase or decreases the value of conformons but the sum of all values of the confomormons present in them is always constant.

## 6 Unbounded Large Value

Starting from the definition of conformons-P system we consider now that conformons present in unbounded copies may have an integer as value. As consequence of this the sum of the values of all the conformons present in the system is unbounded.

We will call *conformons-P system with unbounded value* such variant. The set of numbers generated by these systems considers the conformons present in the final membrane together with their value.

Formally:

$$L(\Pi) = \{[X, x] \mid X \in supp(M_{fin}), (L_1, \cdots, L_m) \Rightarrow^* (M'_1, \cdots, M'_m) \Rightarrow^* \\ \Rightarrow^* (M_1, \cdots, M_m), supp(M'_{ack}) = \emptyset, supp(M_{ack}) \neq \emptyset\}.$$

As stated by the next theorem such a variant of conformons-P system may simulate any program machine.

**Theorem 4.** *The class of numbers generated by conformons-P systems with unbounded value coincides with the one generated by program machines.*

*Sketch of the proof.* Figure 7 represents a conformons-P system with unbounded value simulating a program machine. Each counter present in the simulated machine is represented by a conformon whose value indicates the number stored in the related counter.

The simulation of the test on 0 on a counter is performed choosing nondeterministically to simulate one of the two actions related with it. If the conformons-P system tries to subtract one unit from the value of a counter whose value content is 0, then the simulation process will never reach a final configuration. This will

also happen if the simulation of the state change related to the test on 0 does not see the conformon related to the counter present in a specific membrane.

When the passage of the program machine to a final state is simulated a conformon passes to the acknowledge membrane halting the computation. □

## 7 Final Remarks

It would be interesting to continue to modify the definition of conformons-P systems given in Section 3, removing or adding features, in order to see how this is reflected on the computational power of the systems.

For instance, we can limit to one the kind of labels related to the edges or restrict to a tree the graph underlying a conformons-P system. We may also consider that the interaction between two conformons or the passage of a conformon from one membrane to another may subtract a finite amount from the value of the conformon involved in the operation or that the value of conformons may increase upon entering membranes.

It would be certainly appreciated by the community if one of such variants had an infinite hierarchy on the number of nodes.

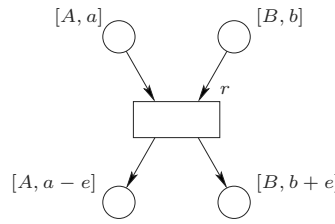



**Fig. 5.** A Petri net representing the interaction of two conformons

Also interesting to investigate is the possibility to model the parallelism present in any variant of conformons-P systems with Petri nets. This kind of nets were introduced by C.A. Petri in his seminal PhD thesis in 1964 to model concurrent and distributed systems. The parallelism, basic in the theoretical facet of biomolecular computing, has not yet been studied and formalized.

The interaction between two conformons defined in (1), may be represented by the Petri net present in Figure 5 where we used the notation indicated in the first chapter of [25].

We think that the possibility to model the parallelism present in conformons-P systems may bring to a better understanding of the behaviour of such systems.

The concept of conformon and the filtering process performed by membranes may be interpreted in a different way than the one described in this paper. The value associated with a conformon may be seen also as the electrical charge, the mass, the size, the momentum, the spin, the speed or frequency of it. The interaction between conformons would allow the passage of one or more of such

features from one conformon to another and membranes might allow the passage of conformons depending on one or more such parameters. Moreover a conformon may be seen not only as associated with a biopolymer but as a generic molecule or a particle akin to a photon or an electron.

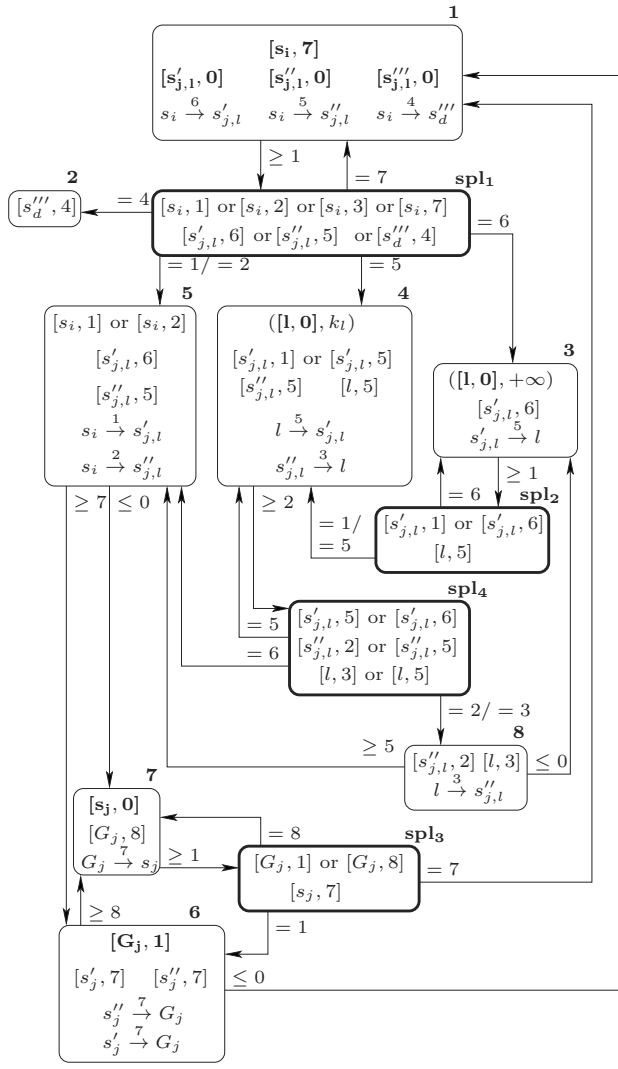



**Fig. 6.** The conformons-P system related to Theorem 2

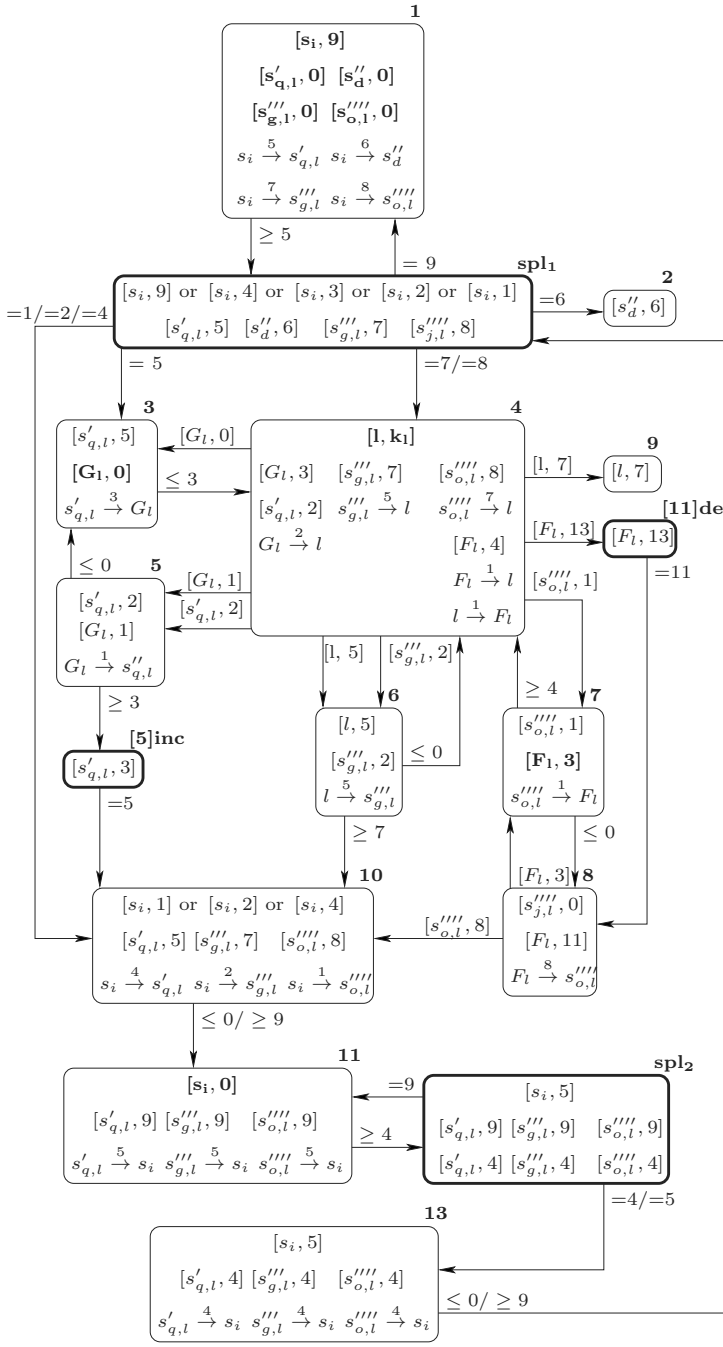



**Fig. 7.** The conformons-P system with infinite value related to Theorem 4

**Acknowledgements**

We thank Hendrik Jan Hoogeboom for the stimulating discussions.

## References

1. B.S. Baker and R.V. Book. Reversal-bounded multipushdown machines. *Journal of computer and system science*, 8:315–332, 1974.
2. J. Castellanos, G. Păun, and A. Rodriguez-Paton. P systems with worm objects. In *IEEE 7th. Intern. Conf. on String Processing and Information Retrieval, SPIRE 2000, La Coruna, Spain*, pages 64–74, 2000. See also CDMTCS TR 123, Univ. of Auckland, 2000 (www.cs.auckland.ac.nz/CDMTCS).
3. P. Frisco and S. Ji. *About conformons-P systems*, 2002. (manuscript in preparation).
4. P. Frisco and S. Ji. Conformons-P systems. In *DNA Computing, 8th international Workshop on DNA-Based Computers, DNA 2002, Sapporo, Japan, 10-13 June 2002*, pages 161–170. Hokkaido University, 2002.
5. S.A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7:311–324, 1978.
6. T. Head. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 49(6):737–759, 1987.
7. H.J. Hoogeboom. Carriers and counters. P-systems with carriers vs. (blind) counter automata. *Developments in Language Theory, DLT2002*, pages 255–268, 2002.
8. S. Ji. A general theory of ATP synthesis and utilization. *Ann. N.Y. Acad. Sci.*, 227:211–226, 1974.
9. S. Ji. The Bhopalator: a molecular model of the living cell based on the concepts of conformons and dissipative structures. *J. theoret. Biol.*, 116:395–426, 1985.
10. S. Ji. *Biocybernetics: a machine theory of biology*, pages 1–237. Molecular theories and cell life and death. Rutgers University Press, New Brunswick, 1991. S. Ji, ed.
11. S. Ji. The cell as the smallest DNA-based molecular computer. *BioSystems*, 52:123–133, 1999.
12. S. Ji. Free energy and information contents of conformons in proteins and DNA. *BioSystems*, 54:107–214, 2000.
13. S. Ji. The Bhopalator: an information/energy dual model of the living cell (II). *Fundamenta Informaticae*, 49(1-3):147–165, 2002.
14. S. Ji and G. Ciobanu. Conformons and biopolymers shape change in cell modelling. accepted in BioSystems, 2002.
15. 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.
16. M. Madhu and K. Krithivasan. P systems with active objects: Universality and efficiency. *MCU, Chisinau, Moldova, Proceedings in LNCS 2055*, pages 276–287, 2001. M. Margenstern and Y. Rogozhin, editors.
17. C. Martin-Vide and V. Mitrana. P systems with valuations. In I. Antoniou, C.S. Calude, and M.J. Dinneen, editors, *Unconventional Models of Computation, UMC'2K, Solvay Institutes, Brussel, 13 - 16 December 2000*, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science., pages 154–166. Springer Verlag, Berlin, Heidelberg, New York., 2000.

18. 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, November 1961.
19. M.L. Minsky. *Computation: Finite and Infinite Machines*. Automatic Computation. Prentice-Hall, 1967.
20. A. Păun, G. Păun, and G. Rozenberg. Computing by communication in networks of membranes. to appear in Intern. J. of Foundations of Computer Science, 2001.
21. G. 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 `http://www.tucs.fi`.
22. G. Păun. Computing with membranes. a variant: P systems with polarized membranes. *Inter. J. of Foundations of Computer Science*, 1(11):167–182, 2000.
23. G. Păun. P systems with active membranes: attacking NP complete problems. *J. Automata, Languages and Combinatorics*, 1(6):75–90, 2001.
24. G. Păun and T. Yokomori. Membrane computing based on splicing. In Erik Winfree and David K. Gifford, editors, *Proceedings 5th DIMACS Workshop on DNA Based Computers, held at the Massachusetts Institute of Technology, Cambridge, MA, USA June 14 - June 15, 1999*, pages 217–232. American Mathematical Society, 1999.
25. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*. Springer Verlag, Berlin, Heidelberg, New York., 1998.

# Accretive Rules in Cayley P Systems

Jean-Louis Giavitto, Olivier Michel, and Julien Cohen

LaMI, umr 8042 du CNRS, Université d'Évry – GENOPOLE
Tour Evry-2, 523 Place des Terrasses de l'Agora
91000 Évry, France
{giavitto,michel,jcohen}@lami.univ-evry.fr

**Abstract.** During a discussion taking place at WMC'01, G. Paun put the question of what could be computed only by moving symbols between membranes. In this paper we provide some elements of the answer, in a setting similar to tissue P systems, where the set of membranes is organized into a finite graph or into a Cayley graph, and using a very simple propagation process characterizing accretive growth. Our main result is to characterize the final configuration as a least fixed point and to establish two series of approximations that converge to it. All the notions introduced (Cayley graph of membranes, accretive rule and iteration) have been implemented in the `MGS` programming language and the two approximation series can be effectively computed in Pressburger arithmetics using the `omega` calculator in the case of Abelian Cayley graphs.

## 1 Introduction

P systems are new distributed parallel computing models based on the notion of membrane structures [Pau99,Pau00,Pau01]. A membrane structure is a nesting of cells represented, e.g., by a Venn diagram without intersection and with a unique superset: the skin. Objects are placed in the areas defined by the membranes and evolve following various transformation rules subject to some conditions: an object can evolve into another object, can pass trough a membrane or dissolve its enclosing membrane, etc. The computation is over for instance when no object can further evolve.

During a discussion taking place at WMC'01, Gheorghe Paun put the question of what could be computed *only by moving objects* between membranes. In its initial presentation, the P systems formalism describes the topology of the membranes as a set of nested regions. Thus, the objects can be viewed as moving between the nodes of the membrane's inclusion tree. Our question can then be rephrased into: "*Starting from a set $S$ of symbols located in a tree, what are the nodes $F$ of the tree that are occupied by symbols after moving them following some rules $\mathcal{R}$?*". With the aim of making this question more precise and amenable to some answers, we made some simplifying assumptions and some generalizations.

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 319–338, 2003.

1. In a first approach, we are only interested by the presence or the absence of symbols in a membrane. Thus, we consider in this paper only one kind of symbols and we can simplify the representation of the content of a membrane as a boolean: *false* means that there are no symbols in the membrane and *true* that there are some.
2. Considering only the presence/absence of symbols on a finite tree is too restrictive: there is only a finite set of observable states for the system. Several natural extensions of the initial formalism consider P systems working on graphs which are not trees [PSY01,MVPPRP01]. In this work, we follow the same way and we consider first *finite graphs* and then *Cayley graphs*. With Cayley graphs we can have a finite presentation for unbound graphs of various shapes. They include rings, grids and more general regular tilings of the plane, etc. They include also (infinite) $n$-ary trees (as the Cayley graph of free groups with $n$ generators), which cover the case of our initial question.
3. Let $\mathrm{D}_0$ be the initial configuration of the P systems. The set $\mathrm{D}_0$ specifies the membranes holding an object at the beginning of the computation. In the case of a finite graph, we obviously consider an arbitrary (finite sub-) set of the membranes. In the case of an infinite Cayley graph $\boldsymbol{G}$, it is also reasonable to handle an infinite starting set, providing that this set is defined in a reasonable (e.g., recursive) way. The idea is that starting from a member $\mathrm{D}_0$ of a canonical family of starting sets, we try to characterize in the same way the resulting set $D$ of nodes occupied after the moves allowed by a set of rules $\mathcal{R}$. In this study, we consider the family of *finite unions of cosets* in $\boldsymbol{G}$.
4. A rule $r \in \mathcal{R}$ specifies if a symbol in a membrane must move to another membrane. According to the assumption 1, we further simplify our problem by considering that the condition of the activation of $r$ must take into account only the presence or the absence of symbols in the neighbor membranes. We name such a rule an *accretive rule*.

Some justifications for this framework are given in the next section. We will show that it constitutes a tractable simplification of a more general process easily programmable in the `MGS` language [GM02]. Moreover, the problem in this form is closely linked with the problem of computing the domain of definition of a systolic function [KMW67,SQ93] or the problem of computing the extension of a data-field [LC94,Gia00].

The rest of this paper is organized as follows. The next section introduces the idea of accretive rules in a P system on a graph. Section 3 formalizes the case of an arbitrary finite graph. The expression of the set of membranes that finally hold an object is given by a polynomial of matrices of bound degree. Section 4 introduces the concepts of Cayley graphs and Cayley P systems and gives a well known example of Cayley P system. Section 5 develops the notion of accretive rule in the framework of the Cayley P systems and the following section introduces the specification of an initial configuration by means of cosets. Section 7 exposes the technical core of our work: we give a formal account of the problem and present some results: we characterize the final set $D$ as a least fixed

point and we construct two series of approximating sets $(\mathrm{D}_n)_{n\in\mathbb{N}}$ and $(\mathrm{E}_n)_{n\in\mathbb{N}}$ such that $\mathrm{D}_n \subseteq D \subseteq \mathrm{E}_n$ and $\lim_{n\to\infty} \mathrm{D}_n = \lim_{n\to\infty} \mathrm{E}_n = D$. Finally, we show how the sets $\mathrm{D}_n$ and $\mathrm{E}_n$ can be effectively computed using Pressburger arithmetics and the `omega` calculator in the case of free Abelian Cayley graphs. Comparisons with previous works and links with similar problems are given in the conclusion.

## 2 Accretive Rules for P Systems on a Graph

The neigborhood relationships of the membranes [GM02] in the case of a "P system on a graph" [PSY01] or a "tissue P system" [MVPPRP01], are defined by the edges of a graph. We consider here an arbitrary graph (e.g., defined by a connection matrix $M$, see figure 1, or by a group presentation, see section 4). We consider only rules fulfilling the following assumptions:

1. an empty membrane may gain an object if some of its neighbors own an object;
2. if a membrane holds an object at time $t$, then the object remains in the membrane at any time $t' > t$.

Condition 1 accounts for a growth process: starting from a defined pattern of already existing objects, a new object is created in an empty place. The growing process does not modify the membranes (e.g., it does not add new membranes) but takes place on empty membranes. The empty membranes which are close to occupied membranes, can be interpreted as the "boundaries" of the organism and such a growing process is termed *accretive growth* [GGMP02] in botanic (where accretive growth is opposed to "intercalary growth" where the growing process is from the inside). Condition 2 ensures a monotonic process: when a membrane has an object, it remains forever. This kind of behavior corresponds to diffusion processes where there is no decay in the substance diffused. Because we consider only the presence or the absence of objects, there is no quantitative account for the diffusion. However such a simplified logical model has proved to be useful, for instance in the modeling of regulatory networks [TTK95] or in tumor growth [YPQ58]. This specific kind of transport rule is very important in morphogenesis and has been studied for a long time under various formalisms (Lindenmayer systems for instance or Eden's model in cellular automata, etc.). They can also be used to model the diffusion of a substance in a tissue.

Several kinds of rules meet assumption 1: for instance, an object can be created in an empty membrane if:

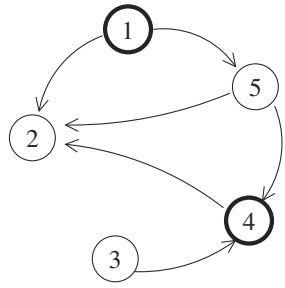
- *all* the neighbors hold a symbol: $\mathcal{R}_{\mathrm{all}}$;
- *one* of the neighbors holds a symbol: $\mathcal{R}_{\mathrm{one}}$;
- *some* of the neighbors hold a symbol: $\mathcal{R}_{\mathrm{some}}$.

The case $\mathcal{R}_{\mathrm{some}}$ subsumes the two other kinds but raises some difficulties in the specification of "some": do we need a precise number of occupied membranes, or only to reach a given number, or a precise set of neighbors... For Cayley graphs,

there is a natural way to define what means "some of the neighbors", and for an arbitrary finite graph, "some" means here "at least one of the predecessor vertices".

## 3 Accretive Rules on a Finite Graph

Let $\boldsymbol{F}$ be a finite graph, specified by its $n \times n$ connection matrix $M = (m_{ij})$ (see Figure 1): $m_{ij} = 1$ if there is an edge from vertex $j$ to vertex $i$. The vertices of this graph are the $n$ membranes of a tissue P systems. The state at time $t$ of the membranes is recorded as a vector $\mathrm{D}_t = (d_i^t)$ where $d_i^t$ takes the value 1 (true) if there is an object in membrane $i$ at time $t$ and else 0 (false).




$$M = \begin{pmatrix} 0&0&0&0&0 \\ 1&0&0&1&1 \\ 0&0&0&0&0 \\ 0&0&1&0&1 \\ 1&0&0&0&0 \end{pmatrix} \qquad \mathrm{D}_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

**Fig. 1.** Accretive rule in a P system on a finite graph. The connection matrix $M$ is a boolean square matrix $(m_{ij})$ where $m_{ij}$ is true if there is an edge between node $j$ and $i$. The membranes (i.e., nodes of this graph) holding an object at the beginning of the computation are in bold and are defined by the initial vector $\mathrm{D}_0$.

*Notations.* Let $\wedge$ denote the logical conjunction and $\vee$ the logical disjunction. Expression $\bar{x}$ denotes the negation of $x$. Let $A = (a_i)$ and $B = (b_i)$ be two boolean vectors of size $n$. The binary operator $\oplus$ extends the disjunction to boolean vectors: $(A \oplus B)_i = a_i \vee b_i$. Let $P = (p_{ij})$ be a $p \times n$ boolean matrix and $Q = (q_{ij})$ be a $n \times q$ boolean matrix. The $\bigotimes_{f,g}$ operator is a generalization of matrix multiplication where binary operation $f$ replaces the multiplication and the binary associative operation $g$ replaces the addition:

$$(P \bigotimes_{f,g} Q)_{ij} = \left( \mathop{g}_{k=1}^{k=n} f(p_{ik}, q_{kj}) \right)_{ij} \qquad \text{for } 1 \leq i \leq p \text{ and } 1 \leq j \leq q.$$

Changing the functions $f$ and $g$ that parameterize $\bigotimes_{f,g}$ allows to take into account several variations of the rules $\mathcal{R}$.

*Specifying $\mathcal{R}_{some}$ Evolution on a Finite Graph.* The effect of a rule $\mathcal{R}_{\text{some}}$ on the state $\mathrm{D}_t$ can be specified by:

$$\mathrm{D}_{t+1} = \varphi(\mathrm{D}_t) = \mathrm{D}_t \oplus M \bigotimes_{\wedge,\vee} \mathrm{D}_t = \begin{pmatrix} d_1^t \vee \bigvee_{k=1}^n m_{1k} \wedge d_k^t \\ \dots \\ d_n^t \vee \bigvee_{k=1}^n m_{nk} \wedge d_k^t \end{pmatrix} \tag{1}$$

For the example in figure 1, we have:

$$\mathrm{D}_1 = \mathrm{D}_0 \oplus M \bigotimes_{\wedge,\vee} \mathrm{D}_0 = \begin{pmatrix} 1 \vee \big((0\wedge1)\vee(0\wedge0)\vee(0\wedge0)\vee(0\wedge1)\vee(0\wedge0)\big) \\ 0 \vee \big((1\wedge1)\vee(0\wedge0)\vee(0\wedge0)\vee(1\wedge1)\vee(1\wedge0)\big) \\ 0 \vee \big((0\wedge1)\vee(0\wedge0)\vee(0\wedge0)\vee(0\wedge1)\vee(0\wedge0)\big) \\ 1 \vee \big((0\wedge1)\vee(0\wedge0)\vee(1\wedge0)\vee(0\wedge1)\vee(1\wedge0)\big) \\ 0 \vee \big((1\wedge1)\vee(0\wedge0)\vee(0\wedge0)\vee(0\wedge1)\vee(0\wedge0)\big) \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

In words: membranes 2 and 5 gain an object at time step 1 because they have a neighbor that holds an object at time step 0.

At each time step, some membranes change their state from 0 to 1. If there are no changes at all, then $\mathrm{D}_{t+1} = \mathrm{D}_t$ and we have reached the final configuration $D = \lim_{n\to\infty} \mathrm{D}_n$. The transition function $\varphi$ is monotonic: the change from 0 to 1 is allowed but the reverse cannot happen. So, after at most $n$ transitions, the final configuration must be reached: $D = \mathrm{D}_n$. This gives us a closed formula for $D$:

$$D = \varphi^n(\mathrm{D}_0) = \mathrm{D}_0 \oplus \bigoplus_{t=1}^{n} (M^t \underset{\wedge,\vee}{\otimes} \mathrm{D}_0) \tag{2}$$

where

$$\begin{aligned} M^1 &= M \\ M^{t+1} &= M \bigotimes_{\wedge,\vee} M^t \qquad \text{for } t \geq 1 \end{aligned}$$

Equation 2 holds because $\bigotimes_{\wedge,\vee}$ is associative and distributive over $\oplus$. This formula is a polynomial of matrices and vectors (although the multiplication is not the standard one) and a lot of properties can be infered by looking at the exponents of the matrix $M$. For instance, one can answer the question "is there an initial configuration $\mathrm{D}_0$ leading to a given final configuration $D$" by solving the polynomial equation $\varphi^n(\mathrm{D}_0) = D$.

## 4 Cayley Shaped Membranes

In this section, we present the notions needed to understand the concept of *Cayley P systems*. A Cayley P systems is a P system on a Cayley graph. Cayley graphs can be infinite, thus allowing P systems on a family of infinite graphs. They also give us an effective tool to give a richer meaning to *some* in the evolution rules of kind $\mathcal{R}_{\text{some}}$ (see section 4.3). We first review some basic facts about Cayley graphs, then we define the Cayley P systems as a special class of P systems on a graph. We conclude this section by an example of Cayley P system.

### 4.1 Finite Group Presentations and Cayley Graphs

A finite presentation $\boldsymbol{G}$ of a group $G$ gives a finite set $\mathtt{G}$ of group generators and a finite set $\mathtt{E}$ of equations constraining the equality of two words. An equation takes the form $v = w$ where $v$ and $w$ are products of generators and of their inverses. We will write $\boldsymbol{G} = \langle \mathtt{G}, \mathtt{E} \rangle$. If $\mathtt{E}$ is empty, it is omitted in the presentation and we say that the group $G$ is free. A given group $G$ admits several presentations and for a given presentation several words can represent the same element in $G$.

We can associate to a presentation $\boldsymbol{G} = \langle \mathtt{G}, \mathtt{E} \rangle$ a labeled graph also denoted by $\boldsymbol{G}$ where:

1. each vertex represents a group element,
2. an edge labeled $s$ is between the nodes $P$ and $Q$ if $P.s = Q$, and
3. the labels of the edges are in $\mathtt{G}$.

This graph is called a *Cayley graph* of $G$. A dictionary between group related notions and graph related notions is illustrated in figure 2.

An Abelian group $G$ is a group where the generators commute. Then every group element $g$ can be written as $\mathtt{g}_1^{x_1}.\mathtt{g}_2^{x_2}\ldots\mathtt{g}_n^{x_n}$ where $\mathtt{g}_i \in \mathtt{G}$ and $x_i \in \mathbb{Z}$. Because we are mainly interested in Abelian groups, we will use an additive notation for the group operation and we will write $g = x_1.\mathtt{g}_1 + x_2.\mathtt{g}_2 + \cdots + x_n.\mathtt{g}_n$ instead. In addition, a group presentation between ⟨ and ⟩ will refer implicitly to an Abelian presentation.

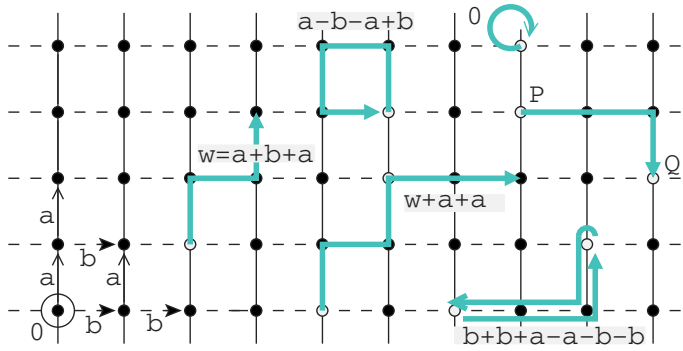



**Fig. 2.** Graphical representation of the relationships between Cayley graphs and group theory. A vertex is a group element. An edge labeled $a$ is a generator $a$ of the presentation. A word (a sum of generators and of inverses of generators in the additive notation) is a path. Path composition corresponds to the group operation. A closed path (a cycle) is a word equal to $e = 0$ (the identity of the addition). An equation $v = w$ can be rewritten $v - w = e$ and then corresponds to a cycle in the graph. There are two kinds of cycles in the graph: the cycles that are present in all Cayley graphs and corresponding to group laws (intuitively: a backtracking path like $b + a - a - b$) and closed paths specific to the own group equations (e.g.: $a - b - a + b$). The graph connectivity (there is always a path going from $P$ to $Q$) is equivalent to say that there is always a solution to the equation $P + x = Q$.

### 4.2 Some Examples of Cayley Graphs

The presentation

$$\mathbf{G2} = \langle\, \texttt{north},\, \texttt{east} \,\rangle$$

is implicitly completed with the equation specifying the commutation of the generators $\texttt{north} + \texttt{east} = \texttt{east} + \texttt{north}$ to define an Abelian group $G2$. The two names `north` and `east` refer to the directions that can be followed to reach the neighbors of an element. So, the corresponding Cayley graph **G2** corresponds to an infinite NEWS mesh with the standard Von Neuman neighborhood (a cell above, below, left or right – not diagonal). The figure 2 represents a part of this graph provided that the generator `a` is renamed in `north` and that `b` is renamed into `east`.

The list of the generators in a presentation can be completed by giving some equations that constraint the displacements in the Cayley graph. For instance,

$$\mathbf{H} = \langle\, \texttt{east},\, \texttt{north},\, \texttt{northeast}\,;\, \texttt{east} + \texttt{north} = \texttt{northeast} \,\rangle$$

defines an hexagonal lattice that tiles the plane, see figure 4. Each cell has six neighbors (following the three generators and their inverses). The equation $\texttt{east} + \texttt{north} = \texttt{northeast}$ specifies that a move following `northeast` is the same as a move following the `east` direction followed by a move following the `north` direction.

Figure 3 gives several other examples of the Cayley graphs of some Abelian and non-Abelian presentations.

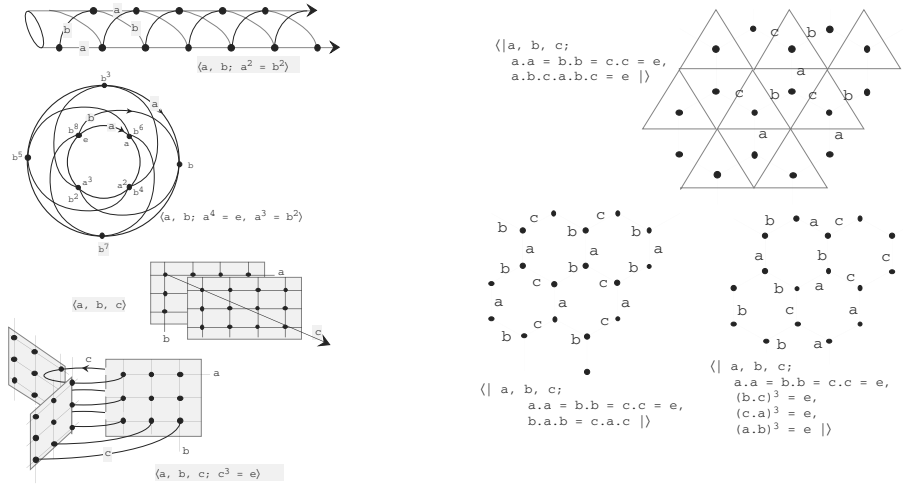



**Fig. 3.** *Left.* Four Abelian group presentations (in multiplicative notation) and their associated graphs. *Right.* Three examples of a 3-neighborhoods shape (multiplicative notation). These triangular neighborhoods are described by non Abelian groups.

### 4.3 Cayley P Systems and Their Rules

A Cayley P system $\mathcal{P}$ is a P system on a Cayley graph $\boldsymbol{G}$. We says that $\boldsymbol{G}$ is the *shape* of $\mathcal{P}$ and the elements $g$ of the group $G$, which are also the vertices of $\boldsymbol{G}$, are called the *membranes* of $\mathcal{P}$. Cayley P systems can be used to specify membranes with a *uniform* neighborhood.

In full generality, the rules of a Cayley P system specify some changes in a membrane content $c$ following some condition $C$. This can be done with rewriting rules of the form:

$$c/C \Rightarrow c'$$

specifying that the content of some membrane $c$ is replaced by $c'$ when the condition $C$ is satisfied.

The condition $C$ may refer to the content of some neighbors. The underlying group structure of a Cayley graph $\boldsymbol{G}$ gives us a convenient tool to speak of the neighbors of a membrane $p$: the neighbors of a membrane $p$ are the membranes $p + \mathtt{g}$ where $\mathtt{g}$ is a generator of $\boldsymbol{G}$. Then, we say that $p + \mathtt{g}$ is a $\mathtt{g}$-neighbor of $p$.

A membrane is identified with a group element, but is also associated to a content. To make a clear distinction between this two associations, we use two distinct notations. If $p$ is a membrane, then $[p]$ represents the content of this membrane and $pos(p)$ refers to the group element that identifies uniquely the membrane. For example, suppose that the membranes hold an integer. Then, a rule like:

$$x/[pos(x) + \mathtt{east}] > 7 \Rightarrow [x] + 1$$

means that the content of a membrane $x$ must be incremented by one if $x$ has an `east`-neighbor with a content greater than 7 (condition: $[pos(x) + \mathtt{east}] > 7$).

A *transformation* $T$ is a set of rules that are applied following a strategy to the membranes of the system to make an evolution step. Usually, a maximal synchronous application strategy is used in P systems. Here we suppose that

- there is no priority between the rules;
- if two rules can apply, one of them is chosen non-deterministically;
- when a rule applies, all the membranes involved in the left hand side of the rule (included in the guard) cannot be used anymore for another rule application in the same time step.

The last constraint expresses that in a time step, the applications of the rules must be independent (i.e., do not share a membrane). If no rules apply it means that there are no membranes matching the left hand side of any rule of $T$. Then the transformation corresponds to the identity function.

### 4.4 An Example of Cayley P System: The Eden Model

Before formalizing the properties of accretive rules in Cayley P systems, let us examine an example in order to clarify the notions and to illustrate the way our systems work.

We consider a simple model of growth sometimes called the Eden model (specifically, a type B Eden model [YPQ58]). The model has been used since the 1960's as a model for such things as tumor growth and growth of cities. In this model, a 2D space is partitioned in empty or occupied cells (we use the value `true` for an occupied cell and `false` for the unoccupied cells). We start with only one occupied cell. At each step, occupied cells with an empty neighbor are selected, and the corresponding empty cell gets occupied.

The Eden's aggregation process on the lattice **G2** is simply described by the following transformation:

$$\begin{aligned}&\texttt{transformation}\ \mathit{Eden} = \{\\ &\qquad x/\mathrm{not}[x] \wedge [\mathit{pos}(x) + \texttt{east}] \Rightarrow \texttt{true}\\ &\qquad x/\mathrm{not}[x] \wedge [\mathit{pos}(x) + \texttt{north}] \Rightarrow \texttt{true}\\ &\qquad x/\mathrm{not}[x] \wedge [\mathit{pos}(x) - \texttt{east}] \Rightarrow \texttt{true}\\ &\qquad x/\mathrm{not}[x] \wedge [\mathit{pos}(x) - \texttt{north}] \Rightarrow \texttt{true}\\ &\}\end{aligned}$$

This transformation is composed of four rules specifying that an empty membrane $x$ (condition: not$[x]$) with a non-empty neighbor following the direction $\mathtt{g}$ (condition: $[\mathit{pos}(x) - \mathtt{g}]$ where $\mathtt{g} \in \{\texttt{east}, -\texttt{east}, \texttt{north}, -\texttt{north}\}$) becomes true (that is, becomes occupied). The rules of an Eden's process on an hexagonal mesh follow the same pattern: there are six rules, one for each generator and one for the inverse of each generator of **H**.

## 5 Accretive Rules in a Cayley P System

Following the remarks of the introduction, we simplify the state of a membrane into a boolean in the context of this work: true means that there is an object in the membrane and false means that the membrane is empty. Thus an accretive rule $f$ in a Cayley P system takes the following form:

$$f \;=\; x/\mathrm{not}[x] \wedge [\mathit{pos}(x) + \mathtt{g}_1] \wedge \ldots \wedge [\mathit{pos}(x) + \mathtt{g}_k] \Rightarrow \mathrm{true} \tag{3}$$

The meaning of such a rule must be obvious: the left hand side matches a membrane $x$ which is empty (predicate not$[x]$ in the condition) and has non-empty neighbors following the directions $\mathtt{g}_1, \ldots, \mathtt{g}_k$.

*The Set of Dependencies of a Rule.* The guard in the left hand side of the rule of eq. 3 accesses to a set of membranes with a well-defined value. We write $\mathrm{R}_f$ for the *set of dependencies* of a rule $f$:

$$\mathrm{R}_f = \{\mathtt{g}_1 \ldots,\ \mathtt{g}_k\}$$

The elements of $\mathrm{R}_f$ are the displacements from an unocuppied membrane to the occupied ones involved by the rule $f$. The *diameter* of $f$ is the number of elements in $\mathrm{R}_f$.

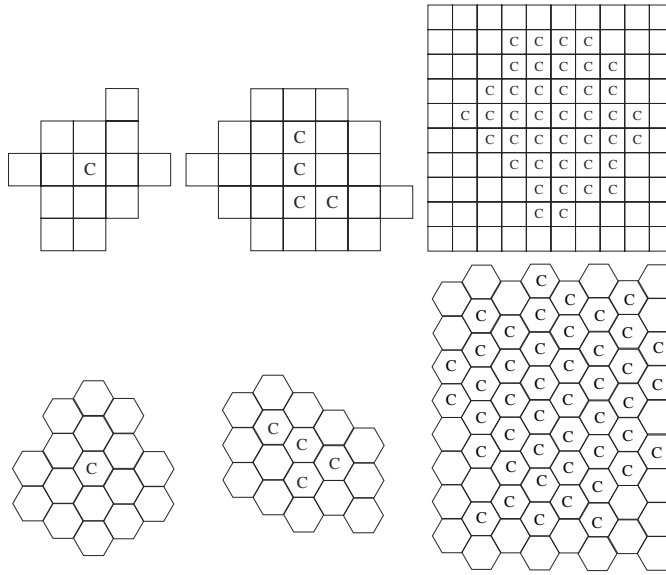


**Fig. 4.** Eden's model on a grid and on an hexagonal mesh (initial state, and states after 2 and 6 time steps). These shapes correspond to a Cayley graph of **G2** and **H** with the following conventions: a vertex is represented as a face and two neighbors in the Cayley graphs share an edge in this representation. Only a part of the infinite domain is figured.

## 6 Configuration of a Cayley P System

As a matter of fact, the state (or configuration) of a Cayley P system $\mathcal{P}$ with the shape $\boldsymbol{G}$ can be represented by a *function* $\pi$ from the group $G$ to a set of values. This function associates to each membrane the multiset of its content in general, and in our special case, a boolean value.

However, for technical reasons, it is more convenient to adopt the following representation: if a membrane $m$ holds the value false, meaning that it does not contain any objects, then $\pi$ is undefined on $m$. Note that $\pi$ is a *partial function*: $\pi : G \to \{\texttt{true}\}$. Thus, specifying $\pi$ is equivalent to the specification of its domain of definition D.

We want to apply iteratively the transformation made of one rule $f$ to some initial configuration $\pi_0$. The first problem is to specify a subset of $G$ corresponding to the domain of definition $\mathrm{D}_0$ of $\pi_0$. Obviously, we want to take into account the underlying group structure and then it is natural to use subgroups of $G$ to specify $\mathrm{D}_0$. We consider then a slight generalization and we will use a *finite union* of *cosets* of $G$ to define $\mathrm{D}_0$.

A coset $u \oplus H = \{u+h, h \in H\}$ in $G$ is the "translation" by $u$ of the subgroup $H$ of $G$. In a non-Abelian group, we have to distinguish the right coset $u \oplus H$ from the left coset $H \oplus u$. To specify a coset we give the element $u$ and the

subgroup $H$. The notation $\langle a_1, a_2, \ldots, a_p \rangle : G$ defines a subgroup of $G$ generated by $\{a_1, a_2, \ldots, a_p\}$ where the $a_i$ are arbitrary elements of $G$. There is no specific equation linking the generators of the subgroup but they are subject to the equations of the enclosing group, if applicable.

An *initial configuration* is a finite union $\mathrm{D}_0$ of cosets $C_1, \ldots, C_p$. The $C_i$ are called the *base cosets* of the Cayley P system. It is easy to have specifically one element at membrane $v$ in an initial configuration: it is enough to have one of the base cosets equal to $v \oplus \langle 0 \rangle : G$ where 0 denotes the neutral element. Therefore, it is possible to describe with $\mathrm{D}_0$ any finite set of membranes.

## 7 Formalization and Results

We restrict ourselves to the transformations with only one rule and we identify the transformation with its only rule. Then, a transformation corresponds to a function $\varphi$ that changes the configuration of the P system.

The Cayley P systems we are interested in are then specified by a triple $(\boldsymbol{G}, \mathrm{D}_0, \varphi)$ where:

- $\boldsymbol{G}$ is a presentation and its Cayley graph corresponds to the membrane's connectivity graph.
- $\mathrm{D}_0$ is the initial configuration and is a finite union of $p$ cosets of the group $G$ specified by the presentation $\boldsymbol{G}$.
- $\varphi$ is the evolution function specified by one evolution rule $f$. The set $\mathrm{R}_f$ of dependencies of the rule $f$ is also denoted by $\mathrm{R}_\varphi$.

If the rule $f$ is specified by equation 3 then the corresponding evolution function $\varphi$ is specified as:

$$\varphi(\pi) = \pi' \quad \text{s.t.} \quad \pi'(p) = \texttt{true} \;\Leftrightarrow\; \pi(p) \;\vee\; \forall \mathtt{g} \in \mathrm{R}_\varphi,\, \pi(p + \mathtt{g}) \tag{4}$$

Note that $\pi(p)$ denotes the fact that $\pi$ is well defined on $p$ and takes the `true` value. We have seen in the previous section that $\pi$ can be identified with its domain of definition D and therefore, the evolution function $\varphi$ can be alternatively defined by its action on the definition domain:

$$\varphi(\mathrm{D}) = \mathrm{D}' \quad \text{s. t.} \quad (p \in \mathrm{D}') \;\Leftrightarrow\; (p \in \mathrm{D}) \;\vee\; \forall \mathtt{g} \in \mathrm{R}_\varphi,\, (p + \mathtt{g}) \in \mathrm{D} \tag{5}$$

In the rest of this paper, we use this last definition of $\varphi$.

Consider an accretive Cayley P system $\mathcal{P} = (\boldsymbol{G}, \mathrm{D}_0, \varphi)$. We call *dependency path* $\ell_p$ of a membrane $p$ with respect to $\mathcal{P}$ a sequence of membranes $p_0, \ldots, p_i, p_{i+1}, \ldots, p_n$ such that: (1) $p_0 = p$; (2) $p_n \in \mathrm{D}_0$, (3) $p_{i+1}$ is a $\mathtt{g}$-neighbor of $p_i$, where $\mathtt{g} \in \mathrm{R}_\varphi$, (4) $p_i \notin \mathrm{D}_0$ for $i \neq n$.

The *trajectory* of $\mathcal{P} = (\boldsymbol{G}, \mathrm{D}_0, \varphi)$ is the sequence $(\mathrm{D}_n)$, $n \in \mathbb{N}$, where $\mathrm{D}_{n+1} = \varphi(\mathrm{D}_n)$. We are looking for a description of the state of the Cayley P system after $t$ evolution steps, that is, we are looking for a closed form of $\mathrm{D}_t$. We are also interested in a closed form of the limit domain

$$D = \lim_{n \to \infty} \mathrm{D}_n$$

### 7.1 An Example

Our problem can be sketched on an example. Figure 5 illustrates the first three iterations of the following rule

$$\begin{aligned} h \;=\; x/ \quad & \mathrm{not}[x] \\ & \wedge\, [pos(x) - \texttt{east} - \texttt{north}] \\ & \wedge\, [pos(x) - 2.\texttt{east}] \\ & \wedge\, [pos(x) - \texttt{north}] \\ & \Rightarrow \texttt{true} \end{aligned} \tag{6}$$

starting from the initial configuration

$$\mathrm{D}_0 = C_1 \cup C_2 \quad \text{with} \quad C_1 = \langle \texttt{east} \rangle : G2 \quad \text{and} \quad C_2 = \langle \texttt{north} \rangle : G2$$

on the shape **G2**.

The integer that appears in a membrane in the right hand side of figure 5 corresponds to the maximal length of a dependency path starting from the membrane and reaching a base coset. This integer can be thought of as the time step when the membrane value is produced (assuming a maximal rule application strategy). In this example, only one value can be produced at each time step. The membranes that have a well-defined value after 3 time steps are drawn as plain square cells.

The infinite path starting from the white dotted membrane $p$ shows the beginning of an infinite sequence of neighbors that must have a value if $p$ holds a value: this path "jumps" over the base cosets and goes to infinity, then, the membrane $p$ cannot have a defined value.

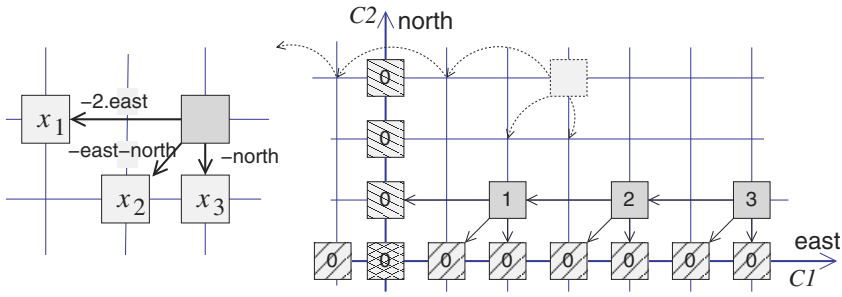



**Fig. 5.** This schema figures a Cayley P system based on a free Abelian shape **G2** = ⟨ `east`, `north` ⟩. The diagram in the left hand side illustrates the set of dependencies of the rule $h$ of eq. 6. The right hand side illustrates the initial configuration of the Cayley P system and the first three iterations of the rule. The initial configuration is defined by the union of two cosets ⟨`east`⟩ : **G2** and ⟨`north`⟩ : **G2**. The dependency set is $\mathrm{R}_h$ = {`-2.east`, `-north`, `-east - north`}. See the text for more explanations.

It should be obvious that

$$\begin{aligned} \mathrm{D}_n &= \mathrm{D}_{n-1} \cup (2n.\mathtt{east} + \mathtt{north}) \oplus \langle 0 \rangle : G2 \\ D &= \mathrm{D}_0 \cup \mathtt{north} \oplus \{2n.\mathtt{east}, n \in \mathbb{N}\} \end{aligned}$$

In the previous expressions, the multiplication $n.x$ of a group element $x$ by a positive integer $n$ denotes the $n$ fold sum $x + \cdots + x$; if $n$ is negative then $n.x$ is the inverse of $(-n).x$.

### 7.2 Characterization of $D$ as a Least Fixed Point

Let $u$ be an element of $\mathrm{D}_n$ such that $u \notin \mathrm{D}_0$, then $u + r_i \in \mathrm{D}_{n-1}$ for all the dependencies $r_i$. Taking the limit in $n$, we have: if $u \in D, u \notin \mathrm{D}_0$, then $u + r_i \in D$. In other words, the set $D$ satisfies the equation

$$D \;=\; \mathrm{D}_0 \cup \bigcap_i D \ominus r_i \tag{7}$$

where $D \ominus r = \{v - r, v \in D\}$. Equation 7 can be rephrased into a fixed point equation $D = \psi(D)$ with the function $\psi$ defined by:

$$\psi = \lambda A.\mathrm{D}_0 \cup \bigcap_i A \ominus r_i$$

The fixpoint equation admits a least solution for the inclusion order (see any standard textbook on domain theory) that can be reached as the limit of $\psi^n(\emptyset)$. We write this solution $\mathrm{fix}(\psi)$. It is immediate to check that $\mathrm{D}_{n+1} \subseteq \psi(\mathrm{D}_n)$, and then we have

$$D = \mathrm{fix}(\lambda A.\mathrm{D}_0 \cup \bigcap_i A \ominus r_i)$$

### 7.3 Definition of the Lower Approximation $\mathrm{D}_n$

Starting from the definition of $\mathrm{D}_n$ we have immediately:

$$\mathrm{D}_0 \subseteq \mathrm{D}_1 \subseteq \ldots \subseteq \mathrm{D}_n \subseteq \ldots \subseteq \mathrm{D}_\infty = D \tag{8}$$

Therefore, the sequence $\mathrm{D}_n$ gives a lower approximation of $D$. Furthermore, it may be remarked that an element $u$ of $\mathrm{D}_{n+1}$ which is not an element of $\mathrm{D}_k, k \leq n$, is such that $(u + r_i) \in \mathrm{D}_n$ for all the dependencies $r_i$. In other word, $u$ belongs to $\bigcap_i (\mathrm{D}_n \ominus r_i)$. We can summarize this result:

$$\mathrm{D}_0 = \bigcup_j C_j \tag{9}$$

$$\mathrm{D}_{n+1} = \mathrm{D}_n \cup \bigcap_i \mathrm{D}_n \ominus r_i \tag{10}$$

### 7.4 The Upper Approximation $\mathrm{E}_n$

*A Geometric Interpretation.* To obtain an upper approximation of $D$, we first interpret geometrically the property of belonging to the domain of definition of $\pi_n$. To each membrane $u \in G$ we associate the set $\mathcal{P}_u$ of directed paths corresponding to the membranes reachable from $u$ using a sequence of dependencies. An element $p$ of $\mathcal{P}_u$ is a word of the *monoid* $\mathcal{R}_\varphi$ generated by $\mathrm{R}_\varphi$:

$$\mathcal{R}_\varphi = \{\ \alpha_1.r_1 + ... + \alpha_k.r_k, \text{ with } r_l \in \mathrm{R}_\varphi \text{ and } \alpha_l \in \mathbb{N}\ \}$$

Note that:

- The dependency paths of $u$ are included in $\mathcal{P}_u$.
- The membrane $u$ cannot be in $D$ if there is a $p \in \mathcal{P}_u$ with an infinite length that does not intersect a base coset.

*Computing an Upper Approximation* $\mathrm{E}_0$. If $u \in D$ is defined, then all the dependency paths starting from $u$ end on a coset $C_j$. Among these paths, some are made only of $r_i$ displacements (for a fixed $i$). Let:

$$\mathcal{R}_i = \{\ -n.r_i,\ n \in \mathbb{N}\ \} \tag{11}$$
$$\mathrm{E}_0 = \mathrm{D}_0 \ \cup \bigcap_i \mathrm{D}_0 \oplus \mathcal{R}_i$$

The set $\mathcal{R}_i$ is the monoid generated by $-r_i$ (*warning:* we take the inverse of the dependency). The expression $A \oplus B$ denotes the set $\{a + b, a \in A, b \in B\}$. The set $\mathrm{E}_0$ is made of the points $u \in G$ that either belong to $\mathrm{D}_0$ or are such that there is a path made only of $r_i$ starting from $u$ and reaching $\mathrm{D}_0$. This last property is simply expressed as: $\forall i, \exists n,\ u - n.r_i \in \mathrm{D}_0$. This property holds for all $u \in D$ and then:

$$D \subseteq \mathrm{E}_0$$

*Refining the Approximation* $\mathrm{E}_0$. The upper approximation $\mathrm{E}_0$ is a little crude. We can refine it on the basis of the following remark. If $u \in D$, then we have either $u \in \mathrm{D}_0$ or $u + r_i \in D$. We can deduce that:

$$D \ \subseteq\ \mathrm{E}_1 = \ \mathrm{D}_0 \ \cup\ (\mathrm{E}_0 \ \cap \bigcap_i \mathrm{E}_0 \ominus r_i)$$

Obviously $\mathrm{E}_1 \subseteq \mathrm{E}_0$. Moreover, this construction starting from $\mathrm{E}_0$ can be iterated, which introduces the sequence

$$\mathrm{E}_0 = \mathrm{D}_0 \ \cup \bigcap_i \mathrm{D}_0 \oplus \mathcal{R}_i \tag{12}$$

$$\mathrm{E}_{n+1} = \ \mathrm{D}_0 \ \cup\ (\mathrm{E}_n \ \cap \bigcap_i \mathrm{E}_n \ominus r_i) \tag{13}$$

We always have $D \subseteq \mathrm{E}_{n+1} \subseteq \mathrm{E}_n$.

Let $\mathrm{E}_\infty$ be the limit of $\mathrm{E}_n$. For each $u \in \mathrm{E}_\infty$, we have either $u \in \mathrm{D}_0$ or $u + r_i \in \mathrm{E}_\infty$. Therefore, $\mathrm{E}_\infty$ is a solution of the equation 7. It should be checked that it is the *least* solution which we admit (intuitively, the element of $G$ are equivalence classes of the *finite* linear combinations of generators and then, if $x \in \mathrm{E}_\infty$ it can be checked by induction on the number of occurrences of $r_i$ in $x$ that $x \in D$).

### 7.5 Summary and a Conjecture

We can summarize the previous results by the formula:

$$\mathrm{D}_0 \subseteq ... \subseteq \mathrm{D}_n \subseteq ... \subseteq \mathrm{D}_\infty = D = \mathrm{E}_\infty \subseteq ... \subseteq \mathrm{E}_n \subseteq ... \subseteq \mathrm{E}_0 \qquad (14)$$

These results can be generalized without difficulty by considering more general base case domains. That is, we may replace the coset $C_i$ by an arbitrary set $S_i$ in equation 9 and relation 14 still holds.

A *monoid* $M$ generated by some elements $u_1, ..., u_p$ of a group $G$ is the set of elements that can be written as a positive linear combination of the $u_i$'s. We call *comonoid* the translation of a monoid, that is, a set $x \oplus M = \{x + m, m \in M\}$ where $M$ is a monoid. For the examples we have worked out on free Abelian groups (that is, for membranes organized into a $d$-dimensional grid and indexed by $\mathbb{Z}$-modules), we have checked that the limit domain $D$ is a *finite union of comonoids*. We conjecture that this is always true.

## 8 Computing the $\mathrm{D}_n$ and $\mathrm{E}_n$ Sets in the Abelian Case

Equations 9, 10, 11, 12 and 13 enable the explicit construction of $\mathrm{D}_n$ and $\mathrm{E}_n$ if we can compute the intersection and the union of *comonoids*.

Indeed, a coset is a special kind of comonoid and the intersection of a comonoid is either empty or a comonoid. If the sum $D \oplus M$ of a comonoid $D$ by a monoid $M$ is also a monoid (which is the case for Abelian shapes or if the $r_i$ commute with all the group elements), then all the arguments of the intersections and unions in the previous equations are comonoids. We may then express $\mathrm{D}_n$ and $\mathrm{E}_n$ for a given $n$ as a finite union of comonoids. It is then clear that the domain of definition $\mathrm{D}_n$ of $\pi_n$ is a finite union of comonoids. The conjecture only says that this union is finite for the limit.

We have used the `omega calculator`, a software package [KMP+96] that enables the computation of various operations on convex polyhedra to do linear algebra in $\mathbb{Z}^n$ and represent comonoids. Linear algebra is not enough to compute $\mathrm{D}_n$ and $\mathrm{E}_n$ because we have to compute the $\mathcal{R}_i$. Fortunately, the `omega calculator` is able to determine in some cases[1] the *transitive closure* of a relation [KPRS94] which enables the computation of $\mathcal{R}_i$ as the transitive closure

[1] We plan to develop a dedicated library under `Mathematica` to compute these approximations systematically.

of the relation `[x, x+r_i]`. We use here the syntax of the `omega calculator` and an expression such as `[`$f$`(x)]`, where `x` is a free variable, denotes the set $\{f(x), x \in \mathbb{Z}\}$ and an expression `[x, `$f$`(x)]`, defines a relation linking `x` to $f$(`x`). Please refer to [KMP+96] for the `omega calculator` concepts and syntax.

Here is in example, based on the Cayley P system illustrated in figure 5. We first define the base cosets in $\mathbb{Z}^2$:

```
C1 := { [n, 0] };
C2 := { [0, n] };
```

then three relations that correspond to the dependencies:

```
r1 := { [x, y] -> [x, y-1] };
r2 := { [x, y] -> [x-2, y] };
r3 := { [x, y] -> [x-1, y-1] };
```

and we need also the inverse of the dependencies:

```
ar1 := { [x, y] -> [x, y+1] };
ar2 := { [x, y] -> [x+2, y] };
ar3 := { [x, y] -> [x+1, y+1] };
```

We may now define the $D_i$:

```
D0 := C1 union C2;

H1 := ar1(D0) intersection ar2(D0) intersection ar3(D0);
D1 := D0 union H1;

H2 := ar1(D1) intersection ar2(D1) intersection ar3(D1);
D2 := D1 union H2;

H3 := ar1(D2) intersection ar2(D2) intersection ar3(D2);
D3 := D2 union H3;
```

We can ask `omega` to compute a representation of $D_3$. The query `D3` returns:

```
{[x,0]} union {[0,y]} union {[4,1]} union {[6,1]} union {[2,1]}
```

which is what is expected. For the approximation $E_i$ we need to represent the monoids $\mathcal{R}_i$ which is done through a transitive closure:

```
AR1 := ar1*;
AR2 := ar2*;
AR3 := ar3*;
```

The definition of $E_0$ raises the computation of

```
E0 := AR1(D0) intersection AR2(D0) intersection AR3(D0);
```

(we have omitted the union with $D_0$ to avoid too complicated terms in the result). The evaluation of this definition returns

```
{[x,y]: Exists (alpha : 0 = x+2alpha && 1 <= y && 2 <= x)}
union {[x,0]} union {[0,y]}
```

This approximation is too large, we may refine it by computing $E_1$:

```
E1 := E0 intersection ar1(E0)
         intersection ar2(E0)
         intersection ar3(E0);
```

The evaluation of $E_1$ gives:

```
{[x,1]: Exists ( alpha : 0 = x+2alpha
                       && 4 <= x)} union {[2,1]}
```

which is also $D$ minus $D_0$.

## 9 Conclusions

In this work, we have considered a specific transport process, called accretive growth, on a set of membranes organized into a Cayley graph. We have formally defined the trajectory of the resulting Cayley P system and characterized the final configuration. We conjecture that this configuration can be described as a finite union of so-called comonoids but this assertion is not proved yet.

The idea to use a group structure to specify the graph underlying a tissue P system has many links with the concepts of GBF developed in the framework of the `MGS` language and the reader may refer to the references cited in the paper. To complete this conclusion, we review two related domains: cellular automata on Cayley graphs and systolic programming.

**Cellular Automata on Cayley Graphs.** Moving symbols between a set of cells is reminiscent of some process described in the cellular automata (CA) literature. Usually, two-dimensionnal CA use a NEWS grid. It is worth mentioning the work of Z. Róka on the extension of the cellular automata formalism to handle more general cell spaces. She considers Cayley graphs in [Rók94,Rók95b,Rók95a] to model both the cell space and the communication links between the cells (the use of Cayley graphs as intersection networks has been extensively studied, see e.g. [Hey97]). This piece of research focuses on the conditions of the simulation of a CA on a given Cayley graph by another CA on another Cayley graph and on the algorithmic problem of the global synchronization of a set of cells.

There are strong links between Cayley P systems and such extensions of cellular automata: in the two cases we have to study the propagation of computations in a space described by a Cayley graph. However, the mentioned work focuses on some synchronization problems and establishes some complexity results for various simulations. For instance, the characterization of the domain is out of the CA scope.

**Systolic Programming.** We recall here the terminology concerning recurrence equations. *Uniform recurrence equations* (URE) have been introduced by Karp, Miller and Winograd [KMW67]. Their model has been broadened to *affine recurrence equations* (ARE). An ARE takes the following form:

$$\forall z \in D, \quad U(z) \;=\; f(U(I(z)),\ V(I'(z)),\ ...) \tag{15}$$

where $D$ is a convex polyhedron of $\mathbb{Z}^n$ called the *domain* of the equation; $z$ is a point of $\mathbb{Z}^n$; $U$ and $V$ are *variable names* indexed by $z$ (the dimension of the index of a given variable is constant). The functions $I, I', \ldots$ are affine mappings from $\mathbb{Z}^n$ to $\mathbb{Z}^n$. The variable $U(I(z))$ is an argument and $U(z)$ is a result of the equation. The function $f$ is strict. If all the mappings $I$ are translations then the system is said to be an URE. This formalism has been largely used. Indeed, there is a large corpus of mathematical results in linear algebra that can help to solve the problems encountered. One of the main problem is to characterize the domain of definition of the function specified by equation 15.

This problem is very similar to the characterization of the limit domain $D$. Indeed, because the function $f$ is strict, the equations defining the domain of definition of $U$ have the same formal expression that the equations defining $D$ (assuming $\mathbb{Z}$ as the underlying group), see [Gia99].

*Domain of Definition of an ARE.* We can review some results in this area. Karp, Miller and Winograd [KMW67] have shown the decidability for URE on a bound domain, without explicitly constructing the dependency graph.

However, B. Joinnault [Joi87] has shown the undecidability when the domain of the equations is not bound. The proof relies on the coding of a Turing machine by an URE. The functions used in the specification of an URE are strict, that is, we do not have a conditional; the conditional is simulated by an adequate specification of the domain of the equations.

This result cannot be adapted in the case of Cayley P systems because the specification of the domain of definition of an URE (which plays for the URE the same role as the base cosets) relies on the specification of convex polyhedra in $\mathbb{Z}^n$ and a convex polyhedron is not generally a coset in $\mathbb{Z}^n$ neither a finite union of cosets or the complementary of a finite union of cosets. In [SQ93], the undecidability result is extended to the case of *parametric bound domains* (i.e., the domain is described by an union of finite convex polyhedron parameterized by a parameter $p \in \mathbb{Z}^m$) (for a given value of $p$, the domain is finite).

Dependencies beyond the affine dependences can be found in exact or approximate data flow analysis [Fea91,**?**]. More specifically for recursive structures, J.-F. Collard and A. Cohen [Coh96] have used the group structure to specify and analyze recursive computations on trees.

**Acknowledgements**

The authors would like to thanks the members of the "Simulation and Epigenesis" group at Genopole for stimulating discussions and biological motivations.

They are also grateful to F. Delaplace and D. Blasco for numerous questions, encouragements and thoughtful remarks. This research is supported in part by the CNRS, the GDR ALP and IMPG, the University of Evry and Genopole/Evry.

## References


Coh96. A. Cohen. Structure de données régulières et analyse de flot. DEA, ENS-Lyon, June 1996.

Fea91. P. Feautrier. Dataflow analysis of scalar and array references. *Int. Journal of Parallel Programming*, 20(1):23–53, February 1991.

GGMP02. J.-L. Giavitto, C. Godin, O. Michel, and P. Prusinkiewicz. *Biological Modeling in the Genomic Context*, chapter "Computational Models for Integrative and Developmental Biology". Hermes, July 2002.

Gia99. J.-L. Giavitto. Scientific report for the tenure. Technical report, LRI, Université de Paris-Sud, centre d'Orsay, September 1999. Research Report 1226.

Gia00. J.-L. Giavitto. A framework for the recursive definition of data structures. In *Proceedings of the 2nd Imternational ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP-00)*, pages 45–55. ACM Press, September 20–23 2000.

GM02. J.-L. Giavitto and O. Michel. The topological structures of membrane computing. *Fundamenta Informaticae*, 49:107–129, 2002.

Hey97. M.-C. Heydemann. *Graph Symmetry*, chapter "Cayley graphs and interconnection networks", pages 167–224. Kluwer Academic Publisher, 1997.

Joi87. B. Joinnault. *Conception d'algorithmes et d'architecture systoliques.* PhD thesis, Thèse de l'Université de Rennes I, September 1987.

KMP+96. W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpeisman, and D. Wonnacott. *The Omega calculator and library, version 1.1.0.* College Park, MD 20742, 18 november 1996.

KMW67. R.M. Karp, R.E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM*, 14(3):563–590, July 1967.

KPRS94. W. Kelly, W. Pugh, E. Rosser, and T. Shpeisman. Transitive closure of infinite graphs and its application. Technical Report UMIACS-TR-95-48, CS-TR-3457, Univ. of Maryland, College Park, MD 20742, 14 Aprils 1994.

LC94. B. Lisper and J.-F. Collard. Extent analysis of data fields. Technical Report TRITA-IT R 94:03, Royal Institute of Technology, Sweden, January 1994.

MVPPRP01. C. Martin-Vide, G. Paun, J. Pazos, and A. Rodriguez-Paton. Tissue P Systems. Technical Report TUCS tech. rep. 421, Turku Centre for Computer Science, September 2001.

Pau99. G. Paun. Computing with membranes: An introduction. *Bulletin of the European Association for Theoretical Computer Science*, 67:139–152, February 1999.

Pau00. G. Paun. Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143, 2000.

Pau01. G. Paun. From cells to computers: Computing with membranes (p systems). *Biosystems*, 59(3):139–158, March 2001.

PSY01. G. Paun, Y. Sakakibara, and T. Yokomori. P systems on graphs of restricted forms. *Publ. Math. Debrecen*, 2001.

Rók94. Z. Róka. One-way cellular automata on Cayley graphs. *Theoretical Computer Science*, 132(1–2):259–290, 26 September 1994.

Rók95a. Z. Róka. The firing squad synchronization problem on CAYLEY graphs. *Lecture Notes in Computer Science*, 969:402–411, 1995.

Rók95b. Z. Róka. Simulations between cellular automata on CAYLEY graphs. *Lecture Notes in Computer Science*, 911:483–493, 1995.

SQ93. Y. Saouter and P. Quinton. Computability of recurrence equations. *Theoretical Computer Science*, 116(2):317–337, August 1993.

TTK95. R. Thomas, D. Thieffry, and M. Kaufman. Dynamical behaviours of regulatory networks - I . biological role of feedback loops and pratical use of the concept of feedback loop. *Bulletin of Mathematical Biology*, 57(2):247–276, 1995.

YPQ58. H.P. Yockey, R.P. Platzman, and H. Quastler, editors. *Symposium on Information Theory in Biology*. Pergamon Press, New York, London, 1958.

# Tissue P Systems with Contextual and Rewriting Rules*

Shankara N. Krishna, Kuppuswamy Lakshmanan, and Raghavan Rama

Department of Mathematics
Indian Institute of Technology Madras
Chennai-600 036, India
ramar@iitm.ac.in

**Abstract.** The study of tissue P systems was initiated in [6], inspired from the way neurons cooperate, processing impulses in the complex net established by synapses. These systems use multisets of objects for processing, and it was shown that computational completeness can be achieved using a small number of cells and states. In this paper, we use string objects as the underlying data structure. The control structure used is a restricted form of contextual rules and rewriting rules. We obtain two characterizations of recursively enumerable languages using these systems: tP systems having 2 states and 2 cells as well as tP systems having 4 states and a single cell generate all recursively enumerable languages. We also discuss the relationships with ET0L and E0L languages.

## 1 Introduction

The P systems are computing models inspired from the structure and the functioning of the living cells [1,8]. A P system consists of a *membrane structure* which is a three dimensional structure of vesicles, all of them placed in a main vesicle, delimited by a *skin* membrane. In the compartments defined by these membranes (vesicles), there are placed multisets of objects. The multisets of objects can be interpreted as chemical compounds swimming in the regions delimited by the membranes. There are evolution rules governing the modification of these objects in each membrane; the objects can also pass through the membranes or leave the system, through the external membrane. In each time unit, all objects in a compartment which can evolve by the rules associated with that compartment have to evolve. In this way, we get *transitions* from a *configuration* to the next one. A sequence of transitions constitutes a *computation*; with each halting computation, we associate a *result*, the number of objects sent out of the system during the computation.

Thus, a P system is a computing device which abstracts from a single cell structure and functioning. But in most cases, since cells live together and are associated with tissues and organs, inter-cellular communication becomes an

* This work was supported partially by a DST Project Sanction No. DST/MS/124/99.

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 339–351, 2003.

essential feature. This communication is done through the protein channels established among the membranes of the neighboring cells [3]. This has been the main motivation for the introduction of tP systems [6].

tP systems are also motivated from the way neurons cooperate. A neuron has a body containing a nucleus; its membrane is prolonged by two classes of fibres: the *dendrites* which form a filamentary bush around the body of the neuron, and the *axon*, a unique, long filament which ends in a filamentous bush. Each of the filaments from the end of the axon is terminated with a small bulb.

Neurons process impulses in the complex net established by *synapses*. A synapse is a contact between an endbulb of a neuron and the dendrites of another neuron. The transmission of impulses from one neuron to another one is done in the following way: A neuron will be *fired* only if it gets sufficient excitation through its dendrites. After firing a neuron and having it excited, there is a small interval of time necessary to synthesize the impulse to be transmitted to the neighboring neurons through the axon; also, there is a small interval of time necessary for the impulse to reach the endbulbs of the axon. The inputs to a neuron can also be from various sensory areas, like the eyes, the skin and the muscles, in addition to the impulses they get from other neurons. The neuron synthesizes an impulse which is transmitted to the neurons to which it is related by synapses; the synthesis of an impulse and its transmission to adjacent neurons are done according to certain *states* of the neuron.

These observations have been made use of while defining a tP system [6]. A tP system consists of several cells, related by protein channels. The term *synapses* is used for referring to these channels. Each cell has a state from a given finite set of states and can process multisets of objects represented by symbols from a given alphabet. The standard rules are of the form $sM \to s'M'$, where $s, s'$ are states and $M, M'$ are multisets of symbols. Some of the elements are marked with an indication "go" and this means they have to leave the cell immediately and pass to the cells to which we have direct links through synapses. This communication can be done in a replicative manner or a non-replicative manner.

In this paper, we consider tissue P systems where the objects are described by strings. The idea of using contextual rules along with rewriting rules was introduced in [5] and a variant of P systems using these kind of rules was investigated in [2]. The control structure used in this paper consists of a restricted form of rewriting rules as well as contextual rules. To a string $z$ in the tP system, we apply contextual rules in the depth-first manner [7]. Similarly, a rewriting rule is applied to a symbol $a$ in $z$ iff it was introduced in the previous step through a contextual rule of the form $(sw, (a, b))$ or $(sw, (b, a))$, or through a rewriting rule $sc \to sa$, where $w$ is a substring of $z$ and $c$ is a symbol in $z$.

As in usual tP systems, we start from an initial configuration, and allow the system to proceed until reaching a halting configuration, where no further rule can be applied. A particular cell is designated as the output cell, and in its rules $sa \to s'x$ or $(sw, s'(u, v))$, where $a \in V, u, v, w, x \in V^*$, the indication "out" is allowed, and such a string is sent out of the system. The terminal strings sent out of the system contribute to the result of the computation.

## 2 Language Theory Prerequisites

In this section, we introduce some formal language theory notions which will be used in this paper; for further details, we refer to [10]. For an alphabet $V$, we denote by $V^*$ the set of all strings over $V$, including the empty one, denoted by $\lambda$. The length of a string $x \in V^*$ (the number of symbol occurrences in $x$) is denoted by $|x|$. The number of occurrences of a given symbol $a \in V$ in a string $x$ is denoted by $|x|_a$. By $CF, CS, RE$ we denote the families of context-free, context-sensitive and recursively enumerable languages, respectively, while $MAT$ denotes the family of languages generated by matrix grammars without appearance checking.

A context-free grammar $G = (N, T, S, P)$ is said to be in the Chomsky normal form if the rules in $P$ are of the form $X \to YZ, X \to a$, where $X, Y, Z \in N, a \in T$. It is known that for every context-free grammar $G$, there exists an equivalent grammar $G'$ in the Chomsky normal form.

A type-0 grammar $G = (N, T, S, P)$ is said to be in Kuroda normal form if the rules in $P$ are of the forms $A \to BC, A \to a, A \to \lambda, AB \to CD$, for $A, B, C, D \in N$ and $a \in T$.

A type-0 grammar $G = (N, T, S, P)$ is said to be in Penttonen normal form if the rules in $P$ are of the following three forms:

1. $X \to \alpha_1\alpha_2$, for $\alpha_1, \alpha_2 \in N \cup T$ such that $X \neq \alpha_1, X \neq \alpha_2, \alpha_1 \neq \alpha_2$,
2. $X \to \lambda$,
3. $XY \to XZ$, for $X, Y, Z \in N$ such that $X \neq Y, X \neq Z, Y \neq Z$.

Now we pass on to defining some basic types of L systems. Basically, a E0L system is a context-free pure grammar with parallel derivations: $G = (V, T, w, R)$, where $V$ is an alphabet, $T \subseteq V$ is the terminal alphabet, $w \in V^*$ (axiom), and $R$ is a finite set of rules of the form $a \to v$ with $a \in V, v \in V^*$, such that for each $a \in V$ there is at least one rule $a \to v$ in $R$ (we say that $R$ is *complete*). For $w_1, w_2 \in V^*$ we write $w_1 \Longrightarrow w_2$ if $w_1 = a_1 \dots a_n, w_2 = v_1 \dots v_n$ for $a_i \to v_i \in R, 1 \leq i \leq n$. The generated language is $L(G) = \{x \in T^* \mid w \Longrightarrow^* x\}$. The family of languages generated by E0L systems is denoted by *E0L*. A tabled E0L system, abbreviated ET0L system, is a system $G = (V, T, w, R_1, \dots, R_n)$, such that each triple $(V, T, w, R_i)$ is an E0L system; each $R_i$ is called a *table*, $1 \leq i \leq n$. The generated language is defined by

$$L(G) = \{x \in T^* \mid w \Longrightarrow_{R_{j_1}} w_1 \Longrightarrow_{R_{j_2}} \dots \Longrightarrow_{R_{j_m}} w_m = x, \\ m \geq 0,\ 1 \leq j_i \leq n,\ 1 \leq i \leq m\}.$$

Each derivation step is performed by rules of the same table. The family of languages generated by ET0L systems is denoted by *ET0L*.

It is known that $CF \subset E0L \subset ET0L \subset CS$. Moreover, *E0L* is incomparable with *MAT*.

In the sequel we will make use of the following normal form for ET0L systems. Each language $L \in ET0L$ can be generated by an ET0L system $G = (V, T, w, R_1, R_2)$ having only two tables. Moreover, from the proof of Theorem

V.1.3 in [9], we see that any derivation with respect to $G$ starts by several steps of $R_1$, then $R_2$ is used exactly once, and the process is iterated; the derivation ends by using $R_2$.

We recall now some basic definitions of contextual grammars. An *internal contextual grammar* is a construct

$$G = (V, A, (S_1, C_1), ...(S_n, C_n)),$$

$n \geq 1$, where

- $V$ is an alphabet,
- $A \subseteq V^*$ is a finite set, called the set of *axioms*,
- $S_i \subseteq V^*,\ 1 \leq i \leq n$, are the sets of *selectors*,
- $C_i \subseteq V^* \times V^*,\ C_i$ finite, $1 \leq i \leq n,$ are the sets of *contexts*.

The usual derivation in the *internal mode* is defined as follows:

$$x \Longrightarrow_{in} y \text{ iff } x = x_1x_2x_3,\ y = x_1ux_2vx_3, \text{ for } x_1, x_2, x_3 \in V^*,$$
$$x_2 \in S_i,\ (u, v) \in C_i, \text{ for some } 1 \leq i \leq n.$$

The language generated by the above grammar $G$ is defined as

$$L_{in}(G) = \{x \in V^* \mid w \Longrightarrow_{in}^* x, \text{ for some } w \in A\}$$

where $\Longrightarrow_{in}^*$ is the reflexive transitive closure of the relation $\Longrightarrow_{in}$.

Given a contextual grammar $G = (V, A, (S_1, C_1), \ldots, (S_n, C_n))$, the *leftmost* derivation with respect to $G$ is given by

$$x \Longrightarrow_{left} y \text{ iff } x = x_1x_2x_3,\ y = x_1ux_2vx_3$$
$$\text{for } x_1, x_2, x_3 \in V^*,\ x_2 \in S_i,\ (u, v) \in C_i,\ 1 \leq i \leq n,$$
$$\text{such that there is no decomposition } x = x_1'x_2'x_3' \text{ with}$$
$$|x_1'| < |x_1| \text{ and } x_2' \in S_j, \text{ for } 1 \leq j \leq n.$$

## 3 Tissue P Systems with String Objects

In this section, we introduce tissue P systems (tP systems) with string objects. A *tissue P system with string objects* of degree $m \geq 1$ (the degree of a system is the number of cells in the system) is a construct

$$\Pi = (O, T, \sigma_1, \ldots, \sigma_m, syn, i_{out}),$$

where:

1. $O$ is a finite non-empty alphabet;
2. $T \subseteq O$ is the terminal or output alphabet;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ (synapses among cells) if $(i, j) \in syn$, then $j$ is a *successor* of $i$ and $i$ is an *ancestor* of $j$;

4. $i_{out} \in \{1, 2, \ldots, m\}$ indicates the *output cell*;
5. $\sigma_1, \ldots, \sigma_m$ are *cells*, of the form

$$\sigma_i = (Q_i, s_{i,0}, L_{i,0}, P_i), 1 \leq i \leq m,$$

where:
(a) $Q_i$ is a finite set of *states*;
(b) $s_{i,0} \in Q_i$ is the *initial state*;
(c) $L_{i,0} \in O^*$ is the set of *initial strings*;
(d) $P_i$ is a finite set of *rules* which can be of one of the following forms:
- A rewriting rule of the form $sa \rightarrow s'y$ or $sa \rightarrow s'(y, tar)$ where $s, s' \in Q_i$, $a \in O$, $y \in O^*$, $tar \in \{go, out\}$, with the restriction that only $P_{i_{out}}$ can contain rules $sa \rightarrow s'(y, out)$.
- A contextual rule of the form $(sw, s'(u, v))$ or $((sw, s'(u, v)), go)$ or $((sw, s'(u, v)), out)$ where $s, s' \in Q_i$, $w \in O^*$, $u, v \in O \cup \{\lambda\}$, with the restriction that only $P_{i_{out}}$ can contain rules of the form $((sw, s'(u, v)), out)$.

Both the above kinds of rules have to be applied in a depth-first manner [4]: (1) when applying rewriting rules, the substring that is rewritten should contain a word that was inserted in the previous derivation either by a rewriting rule or by a contextual rule; (2) when applying contextual rules, the used selector must contain a word that was introduced in the previous derivation by a rewriting rule or a contextual rule.

A tP system as above is said to be cooperative if it contains at least one rewriting rule $sx \rightarrow s'y$ such that $|x| > 1$, and non-cooperative in the opposite case. In this paper, we consider only non-cooperative systems.

When applying a rewriting rule $sa \rightarrow s'y$ to a string $z$, we replace exactly one occurrence of $a$ in $z$ by $y$. In case the rule $sa \rightarrow s'(y, go)$ is applied in a cell $\sigma_i$, then the symbol $a$ is replaced by $y$ and the resultant string $w$ is sent to the cells related by synapses to the cell $\sigma_i$ according to the following modes:

- *repl:* the string $w$ is sent to each of the cells $\sigma_j$ such that $(i, j) \in syn$;
- *one*: the string $w$ is sent to one of the cells (nondeterministically chosen) $\sigma_j$ such that $(i, j) \in syn$.

Similarly, when $sa \rightarrow s'(y, out)$ is applied, the string obtained by replacing $a$ by $y$ is sent out of the system.

If a contextual rule $(sx, s'(u, v))$ is applied to a string $z$, then $u, v$ are inserted on both sides of the substring $x$ of $z$. If we apply $((sx, s'(u, v)), go)$ or $((sx, s'(u, v)), out)$, then the resultant string is sent to the successor cells, in one of the modes considered above, or is sent out.

It is not necessary that a tP system should have both the above kinds of rules. Thus, we have three classes of tP systems: (1) those having only rewriting rules, (2) those having only contextual rules, and (3) those having both kinds of rules. In this paper, we shall consider tP systems having both rewriting rules and contextual rules where the rules are applied according to the following restrictions:

1. We start by applying a contextual rule or a rewriting rule to a string $z \in V^*$ in some cell $\sigma_i$ in a leftmost manner. All the steps following the first step do not have the leftmost restriction.
2. In the next step, the rules are used in the depth-first way, as follows. In short, if a contextual rule is followed by a rewriting rule, then one of the contexts ($\neq \lambda$) inserted in the last step should be rewritten. In case of $\lambda$ contexts, a neighbor (left or right) of $\lambda$ can be rewritten. Similarly, if a contextual rule follows a rewriting rule $sa \rightarrow sy$, then the selector must contain $y$. In case a contextual rule follows another contextual rule, then the selector must contain one of the contexts inserted in the previous step. Similarly, if a rewriting rule follows another rewriting rule, then a symbol introduced by the previous rewriting rule has to be replaced. If a rule $a \rightarrow \lambda$ has been used, then a neighbor (left or right) of $\lambda$ is chosen for rewriting.

Any $m-$tuple of the form $(s_1L_1, \ldots, s_mL_m)$ with $s_i \in Q_i$ and $L_i \in O^*$ is called a configuration of $\Pi$; thus, $(s_{1,0}L_{1,0}, \ldots, s_{m,0}L_{m,0})$ is the initial configuration of $\Pi$. Using the rules from the sets $P_i$, we can define transitions among the configurations of the system. During any transition, some cells can do nothing: if no rule is applicable to the available strings in the current state, then the cell waits until new strings are sent to it from other cells. Each transition lasts one time unit, and the work of the net is synchronized, the same clock marks the time for all cells. A sequence of transitions among the configurations of a tP system is called a computation of $\Pi$. A computation which ends in a configuration where no rule in no cell can be used, is called a halting computation. The language generated by $\Pi$, denoted by $\mathrm{L}_\beta(\Pi)$, is the set of all strings $z \in T^*$ sent out of the system from cell $\sigma_{\mathrm{i_{out}}}$ during a halting computation, $\beta \in \{repl, one\}$. The family of all sets $L_\beta(\Pi)$, generated by all non-cooperative tP systems with at most $m \geq 1$ cells, each of them using at most $r \geq 1$ states, is denoted by $LtP_{m,r}(\beta),\ \beta \in \{repl, one\}$.

**Note 1**: We do not use rules of the form $s \rightarrow s'$ or $s \rightarrow s'w$, or $(s, s'(u, v))$ which are unrealistic from the biological point of view.

**Note 2**: Consider a tP system $\Pi$ having $n$ states $s_1, \ldots, s_n$. Then we say that a state $s_i$ 'switches over' to a state $s_j, j \neq i$, if there exists a rule $s_ia \rightarrow s_jw$ or $(s_ia, s_j(u, v))$. A state $s_i$ is said to have a 'unique switch over' if there exists one and only one $j \neq i$ such that $s_i$ switches over to $s_j$. This means that the rules involving $s_i$ are of one of the following forms:

1. $s_ia \rightarrow s_iw$,
2. $s_ia \rightarrow s_jw, j \neq i$,
3. $(s_ix, s_i(u, v))$ or $((s_ix, s_i(u, v)), go)$ or $((s_ix, s_i(u, v)), out)$,
4. $(s_ix, s_j(u, v))$ or $((s_ix, s_j(u, v)), go)$ or $((s_ix, s_j(u, v)), out)$,

where $a \in O$, $u, v, x \in O^*$, $w \in O^* \cup O^* \times \{go, out\}$. If every state of $\Pi$ has a unique switch over, then we say that $\Pi$ has a unique switch over.

## 4 Preliminary Results

Directly from the definitions we obtain:

**Lemma 1.** *For all* tP *systems* $\Pi$ *where each cell has at most one successor, we have* $L_{repl}(\Pi) = L_{one}(\Pi)$.

The following auxiliary result will be very useful below.

A system with $m$ cells, each of them using $r$ states, is said to be *of type* $(m, r)$.

**Lemma 2.** *(Double Normal Form Lemma) For each system* $\Pi$ *of type* $(n, 1)$ *with only one string in its configurations and at most one successor for each cell, a system* $\Pi'$ *of type* $(1, n)$ *can be constructed, with only one string in each configuration and with a unique switch over, such that* $L_\beta(\Pi) = L_\beta(\Pi')$, $\beta \in \{one, repl\}$. *Also the converse assertion is true (passing from a system of the form of* $\Pi'$ *above to a* $\beta$*-equivalent system of the form of* $\Pi$ *above).*

*Proof.* Let $\Pi = (O, \sigma_1, \ldots, \sigma_n, syn, i_{out})$ be a tP system where each $\sigma_i$ has exactly one state. We construct the tP system $\Pi' = (O, \sigma'_1, \emptyset, 1)$ where $\sigma'_1$ has the states $s_1, s_2, \ldots, s_n$. Each state $s_i$ of $\sigma'_1$ in $\Pi'$ corresponds to the cell $\sigma_i$ of $\Pi$. Both $\Pi$ as well as $\Pi'$ have the same initial string in their initial configurations. Assume that the initial string is present in cell $\sigma_i$ in $\Pi$. Then the initial state of $\Pi'$ is $s_i$.

The rules of $\sigma'_1$ are constructed as follows: Corresponding to each rule $sa \to sw$ or $(sa, s(u, v))$ of $\sigma_i$ in $\Pi$, we have the rule $s_i a \to s_i w$ or $(s_i a, s_i(u, v))$ in $\sigma'_1$ of $\Pi'$. If we have $(i, j) \in syn$ in $\Pi$ and a rule $sa \to s(w, go)$ or $((sa, s(u, v), go))$ in $\sigma_i$, then in $\Pi'$, we have the rules $s_i a \to s_j w$ or $(s_i a, s_j(u, v))$.

In this way, we simulate the work of each cell $\sigma_i$ by using the corresponding state $s_i$. If a string is sent out of $\Pi$ by a rule $sa \to s(w, out)$ or $((sa, s(u, v), out)$ in $\sigma_i$, the in $\Pi'$, we have the rules $s_i a \to s_i(w, out)$ or $((s_i a, s_i(u, v)), out)$.

Clearly, $L_\beta(\Pi) \subseteq L_\beta(\Pi')$, $\beta \in \{one, repl\}$.

To prove the reverse inclusion, construct a tP system $\Pi$ having $n$ cells from a system $\Pi'$ having one cell and $n$ states. If there is a rule that switches the state from $s_i$ to $s_j$ in $\Pi'$, then the synapse $(i, j)$ is included in $\Pi$. The rules of $\Pi$ are constructed by reversing the procedure that we adopted above. Clearly, $L_\beta(\Pi') \subseteq L_\beta(\Pi)$, $\beta \in \{one, repl\}$, and hence, we have the equality. □

In view of this lemma, all assertions which involve a family $LtP_{1,m}(\beta)$ are also true for the family $LtP_{m,1}(\beta)$ and conversely. In all the proofs below, our systems have only one string inside them at any point of time.

## 5 The Computing Power

**Theorem 1.** $LtP_{2,2}(\beta) = RE$, $\beta \in \{one, repl\}$.

*Proof.* We prove only the inclusion $\subseteq$, the opposite one can be obtained as a consequence of the Turing-Church thesis. Consider a type-0 grammar $G = (N, T, S, P)$ in the Pentonnen normal form. Construct the tP system

$$\Pi = (O, T, \sigma_1, \sigma_2, (1, 2), (2, 1), 1)$$

with the alphabet

$$O = N \cup T \cup \{C', F, F', F'' \mid C, F \in N\},$$

and the following cells:

$$\begin{aligned}
\sigma_1 = (&\{s, s'\}, \{s\}, \{S\},\\
&\{sS \to sx \mid S \to x \in P\} \cup \{(sx, s(\lambda, \lambda)) \mid x \in (N \cup T)^*,\ |x| = 2, 0\}\\
&\cup \{sA \to sy \mid A \to y \in P\} \cup \{((sAB, s'(\lambda, C')), go) \mid AB \to AC \in P\}\\
&\cup \{s'F' \to s(F, go) \mid F \in N\} \cup \{((sa, s(\lambda, \lambda)), out) \mid a \in T\}),\\
\sigma_2 = (&\{s, s'\}, \{s\}, \{\lambda\},\\
&\{(sF', s'(\lambda, F'')) \mid F \in N\} \cup \{s'B \to s(\lambda, go) \mid B \in N\}\\
&\cup \{sF'' \to s(\lambda, go)\} \cup \{(sF, s(\lambda, \lambda)) \mid F \in N\}).
\end{aligned}$$

In the initial configuration, we have the start symbol $S$ in cell 1. A rule of the form $S \to x$ is simulated by the rule $sS \to sx$. Similarly, rules of the form $A \to y$ are simulated by $sA \to sy$. A rule can be applied to a symbol $A$ anywhere in the string by suitably using the rule $(sx, s(\lambda, \lambda))$. For simulating the rule $AB \to AC \in P$, we first apply $((sAB, s(\lambda, C'), go))$. This results in a string $x\underline{\lambda}AB\underline{C'}y$, where $x, y \in (N \cup T)^*$ which is sent to cell 2 (the underlined symbols indicate that they are inserted in this derivation step). In cell 2, the applicable rules are $(sF', s'(\lambda, F''))$ and $(sF, s(\lambda, \lambda))$. The application of the second rule changes nothing. When the first rule is applied, the state $s$ switches to $s'$ and the string becomes $xAB\underline{\lambda}C'\underline{C''}y$. The only applicable rule is now $s'B \to s(\lambda, go)$ which deletes the $B$ and takes the resultant string $xA\underline{\lambda}C'C''y$ to cell 1. The current state in cell 1 is $s'$, and the rule $s'F' \to s(F, go)$ is applied. This brings the string $xA\underline{C}C''y$ in cell 2. In cell 2, the rule $(sF, s(\lambda, \lambda))$ is applied, followed by $sF'' \to s(\lambda, go)$. This brings the string $xACy$ in cell 1. The string can leave the system at any point of time by applying the rule $((sa, s(\lambda, \lambda)), out)$ in cell 1. The terminal strings which leave the system are listed in the language. □

**Theorem 2.** $LtP_{1,4}(\beta) = LtP_{4,1}(\beta) = RE,\ \beta \in \{one, repl\}$.

*Proof.* We prove the result for tP systems of type (1, 4). The result for the other case follows from Lemma 2.1.

Consider a type-0 grammar $G = (N, T, S, P)$ in Kuroda normal form. We construct the tP system

$$\Pi = (O, T, \sigma_1, \emptyset, 1)$$

where:

$$
\begin{aligned}
&O = N \cup T \cup \{C_D \mid C, D \in N\} \cup \{D' \mid D \in N\},\\
&\sigma_1 = (\{s, s', s'', s'''\}, \{s\}, \{S\},\\
&\qquad \{sS \to sx \mid S \to x \in P,\ |x| = 0, 1, 2\} \cup \{sA \to sy \mid A \to y \in P\}\\
&\quad \cup \{(sx, s(\lambda, \lambda)) \mid\ |x| = 0, 1, 2\} \cup \{(sAB, s'(\lambda, C_D)) \mid AB \to CD \in P\}\\
&\quad \cup \{(s'C_D, s''(\lambda, D'))\}, \{s''B \to s'''\} \cup \{s'''A \to s \mid A, B \in N\}\\
&\quad \cup \{sC_D \to sCD, sD' \to s\} \cup \{((sa, s(\lambda, \lambda)), out) \mid a \in T\}).
\end{aligned}
$$

As in Theorem 1, we have the start symbol $S$ in cell 1. Rules of the form $A \to y,\ |y| \leq 2$, can be simulated as in the above theorem. For simulating a rule $AB \to CD$, we apply the rule $(sAB, s'(\lambda, C_D))$ to a string $xABy, x, y \in (N \cup T)^*$. This results in a string $x\underline{\lambda}AB\underline{C_D}y$. Then, the rule $(s'C_D, s''(\lambda, D'))$ is applied resulting in a string $xAB\underline{\lambda}C_D\underline{D'}y$. Next, the rules $s''B \to s''', s'''A \to s$ are applied resulting in the string $x\underline{\lambda}C_DD'y$. This is followed by applying the rules $sC_D \to sCD$ and $sD' \to s$, and we obtain the string $xCDy$.

In this way, it is possible to simulate any rules of $P$ in $\Pi$. The string can leave the system by applying the rule $(sa, s(\lambda, \lambda)), a \in T$. Consequently, $L_\beta(\Pi)$ consists of all terminal strings of $L(G)$ sent out of $\Pi$. □

**Theorem 3.** *$LtP_{3,1}(one) - ET0L \neq \emptyset$.*

*Proof.* Construct the tP system

$$\Pi = (\{\#_1, a, b, c, Z, k, \#_2\}, \{a, c\}, \sigma_1, \sigma_2, \sigma_3, \{(1, 2), (2, 3), (3, 2), (3, 1)\}, 3)$$

with the following cells:

$$
\begin{aligned}
&\sigma_1 = (\{s\}, \{s\}, \{\#_1 a \#_2\}\\
&\quad \cup \{(s\#_1, s(\lambda, \lambda)), sk \to sZ, sZ \to sZ\}\\
&\quad \cup \{sa \to sbb, (sbb, s(\lambda, \lambda)), (sc, s(\lambda, \lambda)), ((s\#_2, s(\lambda, \lambda)), go)\}),\\
&\sigma_2 = (\{\{s\}, \{s\}, \{\lambda\},\\
&\qquad \{(sb, s(\lambda, \lambda)), (sc, s(\lambda, \lambda)), (sa, s(\lambda, \lambda))\}\\
&\quad \cup \{sb \to s(ac, go), sb \to s(ca, go), sk \to s(\lambda, go)\}),\\
&\sigma_3 = (\{\{s\}, \{s\}, \{\lambda\},\\
&\quad \cup \{(sac, s(k, \lambda)), (sca, s(k, \lambda)), sb \to sa\}\\
&\quad \cup \{(sc, s(\lambda, \lambda)), (sa, s(\lambda, \lambda))\}\\
&\quad \cup \{((sc\#_2, s(\lambda, \lambda)), go), ((sa\#_2, s(\lambda, \lambda)), go)\}\\
&\quad \cup \{(sc\#_2, s(\lambda, \lambda)), (sa\#_2, s(\lambda, \lambda))\}\\
&\quad \cup \{((s\#_1 a, s(\lambda, \lambda)), go), ((s\#_1 c, s(\lambda, \lambda)), go)\}\\
&\quad \cup \{(s\#_1 a, s(\lambda, \lambda)), (s\#_1 c, s(\lambda, \lambda))\}\\
&\quad \cup \{s\#_2 \to s(\lambda, go), s\#_1 \to s(\lambda, out)\}).
\end{aligned}
$$

To begin with, we have the string $\#_1 a \#_2$ in cell 1. The rule $sa \to sbb$ as well as the contextual rules in cell 1 double the number of $a's$ by $b's$. When the right end marker is reached, the string is sent to cell 2. In cell 2, we replace exactly one occurrence of $b$ by $ac$ or $ca$ and send the resultant string to cell 3. In cell 3, the rule $(sac, s(k, \lambda))$ or $(sca, s(k, \lambda))$ is applied first. Then, the rules $sb \to sa$ are applied to the right of $ac$ or $ca$. When the right end marker is reached, the string is sent to cell 2. If the string is sent to cell 1, two things can happen: (1) the computation loops due to the introduction of the trap symbol $Z$ by the rule $sk \to sZ$, or (2) the rule $((s\#_2, s(\lambda, \lambda)), go)$ is applied, which takes the string to cell 2.

In cell 2, the rules $(sa, s(\lambda, \lambda)), (sc, s(\lambda, \lambda))$ are applied from the right end marker till $k$ is reached. Then the rule $sk \to s(\lambda, go)$ is applied. The string comes back to cell 3 after deleting $k$. Now the rule $sb \to sa$ is applied to all the $b$'s to the left of $k$, till the left end marker is reached. After this, the computation can be continued or the string can be sent out of the system by applying the rules $((s\#_1 a, s(\lambda, \lambda)), go), ((s\#_1 c, s(\lambda, \lambda)), go)$, or $s\#_1 \to s(\lambda, out)$.

Note that the string sent out by applying the rule $s\#_1 \to s(\lambda, out)$ will be accepted only if the rule $s\#_2 \to s(\lambda, go)$ has been applied previously in cell 3. Consequently, the language generated by $\Pi$ is $\{x \mid |x|_a = 2^{|x|_c},\ |x|_c \geq 1\}$, which is not in $ET0L$. □

**Theorem 4.** $ET0L \subset LtP_{3,1}(one)$.

*Proof.* We prove the inclusion $\subseteq$ here, its strictness follows from the above theorem.

Let $G = (V, T, w, R_1, R_2)$ be an ET0L system in the normal form. Each rule in the tables $R_1, R_2$ is of the form $a \to x_1 x_2 \dots x_n$ where $a, x_i \in V$ and $n \geq 1$. Consider all rules $a \to x_1 x_2 \dots x_n$ in $R_1$ where $n > 2$. We replace each such rule by a sequence of rules $sa \to sx_1 D_1, sD_1 \to sD_2, \dots, sD_{n-2} \to sx_{n-1} x_n$, where $D_i, 1 \leq i \leq n-2$ are new symbols. Let $V_1$ represent the set of these new symbols. Similarly, let $V_2$ denote the set of new symbols used in a similar procedure for the rules of $R_2$.

Now, all rules in each $R_i$ are of the form $a \to x$, where $a \in V \cup V_i, x \in (V \cup V_i)^*,\ |x| \leq 2$.

Let $h$ be the morphism defined by $h(a) = a'$, for all $a \in V$. Construct a tP system

$$\Pi = (O, T, \sigma_1, \sigma_2, \sigma_3, \{(1,2), (1,3), (2,1), (3,1)\}, 3),$$

where:

$$\begin{aligned}
O &= V \cup V_1 \cup V_2 \cup \{Z, k, l\} \cup \{h(a) \mid a \in V\},\\
\sigma_1 &= (\{s\}, \{s\}, \{\#_1 h(w) \#_2\},\\
&\quad \{(s\#_1, s(\lambda, \lambda))\} \cup \{sa' \to slx'lD \mid a \to xD \in R_1, D \in V_1\}\\
&\quad \cup \{(slx'lD, s(\lambda, \lambda)) \mid D \in V_1\} \cup \{sa' \to s \mid a \to \lambda \in R_1\}\\
&\quad \cup \{sa' \to slx'l \mid a \to x \in R_1, x \in V^*\} \cup \{(slx'l, s(\lambda, \lambda)) \mid x \in V^*\}
\end{aligned}$$

$$\cup\{((s\#_2, s(\lambda,\lambda)), go), s\#_2 \to s(\lambda, go)\} \cup \{sa \to sZ \mid a \in V\}$$
$$\cup\{sD \to sZ \mid D \in V_2\} \cup \{sZ \to sZ, sk \to s\}),$$
$$\sigma_2 = (\{s\}, \{s\}, \{\lambda\},$$
$$\{(s\#_2, s(\lambda,\lambda)), sl \to s\} \cup \{sD \to sZ \mid D \in V_1\} \cup \{sZ \to sZ\}$$
$$\cup\{(sa', s(\lambda,\lambda)) \mid a \in V\} \cup \{((s\#_1, s(\lambda,\lambda)), go)\}),$$
$$\sigma_3 = (\{s\}, \{s\}, \{\lambda\},$$
$$\{sl \to s\} \cup \{sa' \to skx'k \mid a \to x \in R_2, x \in V^*\}$$
$$\cup\{(skx'k, s(\lambda,\lambda)) \mid x \in V^*\} \cup \{sa' \to sx \mid a \to x \in R_2, x \in (V \cup V_2)^*\}$$
$$\cup\{(sx, s(\lambda,\lambda)) \mid x \in (V \cup V_2)^*,\ |x| \leq 2\}$$
$$\cup\{sa' \to skx'kD \mid a \to xD \in R_2, D \in V_2\}$$
$$\cup\{(skx'kD, s(\lambda,\lambda)) \mid D \in V_2\} \cup \{((s\#_1 kx'k, s(\lambda,\lambda)), go) \mid x \in V^*\}$$
$$\cup\{(s\#_1 a, s(\lambda,\lambda)) \mid a \in V\} \cup \{s\#_1 \to s(\lambda, out)\}$$
$$\cup\{sD \to sZ \mid D \in V_1\} \cup \{sZ \to sZ\}).$$

In the initial configuration, we have the string $\#_1 h(w)\#_2$ in cell 1, where $w$ is the axiom of $G$. Table 1 is simulated in cell 1 using the rules $sa' \to slx'l$, $sa' \to sly'lD$ corresponding to rules $a \to x$, $a \to yD$, where $a, y \in V$, $x \in V^*$, and $D \in V_1$. Once the right end marker is reached after rewriting all symbols, the string is sent either to cell 2 or to cell 3. Note that if any symbol $D \in V_1$ remains in the string while being sent to cells 2 or 3, then the computation will loop due to the application of the rule $sD \to sZ$ in these cells.

In cell 2, the symbols $l$ are eliminated using the rule $sl \to s$ and the resulting string is sent back to cell 1 for simulating table 1 again. In cell 3, the simulation of table 2 is done starting from the right end marker, eliminating all the $l$'s. From cell 3, we have two choices: either to continue the computation by simulating table 1, or to stop the computation and send the string out of the system. To achieve the former, rules of the kind $sa' \to skx'k, sa' \to sky'kD$ are applied corresponding to rules $a \to x, a \to yD$ in $R_2$, where $a, y \in V$, $x \in V^*$, and $D \in V_2$. The string is then sent to cell 1 by applying $((s\#_1 kx'k, s(\lambda,\lambda)), go)$. In cell 1, table 1 is simulated and the $k$'s are eliminated. Note that the presence of symbols $D \in V_2$ in the string that is being sent to cell 1 will loop the computation, since the rule $sD \to sZ$ will be applied in cell 1.

To stop the computation, the rules $sa' \to sx, x \in (V \cup V_2)^*$, are applied. When the left end marker is reached, the rule $s\#_1 \to s(\lambda, out)$ is applied sending a string over $V^*$ out of the system. Note that the rule $s\#_2 \to s(\lambda, go)$ must be applied in cell 1 while sending the string to cell 3 in case of halting the computations; otherwise, the string that is sent out will not be listed in $L_{one}(\Pi)$. Consequently, $L_{one}(\Pi) = L(G)$. □

**Theorem 5.** $LtP_{1,2}(\beta) - E0L \neq \emptyset,\ \beta \in \{one, repl\}$.

*Proof.* Construct the tP system

$$\Pi = (\{a, b\}, \{a, b\}, \sigma_1, \emptyset, 1)$$

with the following rules:

$$\begin{aligned}\sigma_1 = (&\{s, s'\}, \{s\}, \{aba\},\\ &\{(saa, s(\lambda,\lambda)), (sba, s(\lambda,\lambda)), (sab, s(\lambda,\lambda))\}\\ &\cup \{(sbb, s(\lambda, b)), (sbb, s(\lambda,\lambda)), (sab, s'(a, b))\}\\ &\cup \{(s'bb, s'(\lambda,\lambda)), (s'bb, s'(\lambda, b)), (s'ba, s(\lambda, a))\}\\ &\cup \{((s'ba, s(\lambda, a)), out), ((sbb, s(\lambda, b)), out), ((sbb, s(\lambda,\lambda)), out)\}\\ &\cup \{sb \to sbb, s'b \to s'bb\}).\end{aligned}$$

Clearly, the language generated by $\Pi$ is $\{a^m b^n a^m \mid n \geq m \geq 1\}$, and this language is not in $E0L$. □

**Theorem 6.** $E0L \subset LtP_{1,2}(\beta),\ \beta \in \{one, repl\}$.

*Proof.* We prove only the inclusion here, the strictness follows from the above theorem.

Let $G = (V, T, w, P)$ be an E0L system, the rules of $P$ being of the form $a \to x, a \in V,\ x \in V^*$. As in Theorem 4, we shall replace each rule $a \to x_1 x_2 \ldots x_n, n > 2$, by rules of the form $a \to x_1 D_1, \ldots D_{n-2} \to x_{n-1} x_n$. Let $V'$ denote the set of the newly introduced symbols. Now, the rules in $P$ are of the form $a \to x$, where $a \in V \cup V',\ x \in (V \cup V')^*,\ |x| \leq 2$.

Construct the tP system

$$\Pi = (V \cup V' \cup \{l\}, T, \sigma_1, \emptyset, 1)$$

with the following rules:

$$\begin{aligned}\sigma_1 = (&\{s, s'\}, \{s\}, \{\#_1 w \#_2\},\\ &\{(s\#_1, s(\lambda,\lambda))\} \cup \{sa \to slxl \mid a \to x \in P, x \in V^*\}\\ &\cup \{(slxl, s(\lambda,\lambda)) \mid x \in V^*\} \cup \{sa \to slxlD \mid a \to xD \in P, D \in V'\}\\ &\cup \{(slxlD, s(\lambda,\lambda)) \mid D \in V'\} \cup \{(s\#_2, s'(\lambda,\lambda))\}\\ &\cup \{s\#_2 \to s'\} \cup \{(s'a, s'(\lambda,\lambda)) \mid a \in V\} \cup \{s'l \to s'\}\\ &\cup \{(s'\#_1, s(\lambda,\lambda))\} \cup \{(s'\#_1 \to s(\lambda, out))\}).\end{aligned}$$

The simulation of the rules of $P$ are done in state $s$ as in Theorem 4. Once the right end marker is reached, the state $s$ becomes $s'$. The string is traversed back in this state eliminating the $l$'s till the left end marker is reached. If the rules $s\#_2 \to s'$ and $s'\#_1 \to s(\lambda, out)$ are applied at the right and left ends of a string $\#_1 z \#_2$ while traversing from right to left, in $|z + 2|$ steps, and if the string sent out contains only terminals, then it will be listed in $L_\beta(\Pi),\ \beta \in \{one, repl\}$. Consequently, $L_\beta(\Pi) = L(G)$. □

**Corollary 1.** $LtP_{2,1}(\beta) - MAT \neq \emptyset,\ \beta \in \{one, repl\}$.

*Proof.* The proof follows from the fact that $E0L - MAT \neq \emptyset$. □

## 6 Conclusion

We have investigated the power of tissue P systems working with string objects using a small number of cells (at most four). The systems we have considered here have used a restricted form of contextual and rewriting rules. It should be noted that we have used only finite selectors in all the contextual rules. The problem of whether the family $MAT$ is contained in $LtP_{2,1}(\beta)$, $\beta \in \{one, repl\}$, is left open. It is also worthwhile to investigate the possibility of solving hard problems using these systems in the *repl* mode.

## References


1. C.S. Calude and Gh. Păun, *Computing with Cells and Atoms: An Introduction to Quantum, DNA and Membrane computing*, Taylor & Francis, London, 2001.
2. S.N. Krishna, K. Lakshmanan, and R. Rama, Hybrid P Systems, *Romanian Journal of Information Science and Technology*, **4**, 1-2 (2001), 111–123.
3. W.R. Loewenstein, *The Touchstone of Life. Molecular Information, Cell Communication, and the Foundations of Life,* Oxford Univ. Press, New York, Oxford, 1999.
4. C. Martin-Vide, J. Miquel-Verges, and Gh. Păun, Contextual Grammars with Depth-first Derivation, *Tenth Twente Workshop on Language Technology; Algebraic Methods in Language Processing*, Twente, 1995, 225–233.
5. C. Martin-Vide, A. Mateescu, and Gh. Păun, Hybrid Grammars: The Chomsky-Marcus Case, *Bulletin of EATCS*, **64** (Febr. 1998), 159–165.
6. C. MartinVide, Gh. Păun, J. Pazos, and A. Rodriguez Paton, Tissue P Systems, *Turku Center for Computer Science-TUCS Report* No **421**, www.tucs.fi, Sept. 2001.
7. Gh. Păun, *Marcus Contextual Grammars*, Kluwer Academic Publishers, 1997.
8. Gh. Păun, Computing with Membranes, *Journal of Computer and System Sciences*, **61**, 1 (2000), 108–143.
9. G. Rozenberg and A. Salomaa, *The Mathematical Theory of L Systems*, Academic Press, New York, 1980.
10. G. Rozenberg and A. Salomaa, eds., *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Berlin, 1997.

# Considerations on a Multiset Model for Membrane Computing

Manfred Kudlek[1] and Victor Mitrana[2]

[1] Fachbereich Informatik, Universität Hamburg
D-22527 Hamburg, Vogt-Kölln-Str. 30, D-22527 Hamburg, Germany
kudlek@informatik.uni-hamburg.de
[2] Facultatea de Matematică, Universitatea din Bucureşti
Str. Academiei 14, 70109, Bucureşti, Romania
mitrana@funinf.cs.unibuc.ro, vmi@fll.urv.es

**Abstract.** We define, in an inductive way, structured multisets, as well as Chomsky–like productions for such higher order multisets. The usefulness of the approach for Petri nets and membrane computing is briefly discussed.

## 1 Introduction

In [4,5,6] *P systems* were introduced as a new paradigm for computing, namely computing with membranes. The main idea is that productions together with a (simple) multiset are inside a cell bordered by a membrane. The entire cell structure is hierarchical. The productions are context-independent and have to be applied in a (not necessarily total) parallel way inside a cell (some 0L way) on members of the multiset. The membrane can also be dissolved by some special productions with all productions dissapearing and leaving the multiset inside the outer region. Applications in different cells are in parallel. It is also possible to move multisets to another cell in the cell structure. Also sequential ways have been considered [1].

Membrane systems (cells with productions, P systems) can also be seen as multisets of a more complex structure, especially in a hierarchical way. Thus P systems of P systems, etc. may be considered. Also the problem of existence and construction of a universal structure of such kind arises, in particular not of the power of Turing machines.

In [3], Chomsky-like grammars have been considered for multiset rewriting, and a hierarchy of generative power has been presented. The multisets are simple, i.e., multisets over an alphabet $\Sigma$. Such multiset grammars describe the behaviour of *Petri nets*, especially the reachability sets of simple tokens in simple *place/transition* nets. To enhance the descriptive power of Petri nets, *coloured* Petri nets have been introduced where the tokens are structured objects, even Petri nets of some kind. A still controversally discussed problem is the relation between the firing of transitions in the higher level net and the lower level nets (the objects). Among others, *reference* and *value* semantics have been introduced.

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 352–359, 2003.

Another problem is to find a universal Petri net of higher order, simulating all other Petri nets of some kind, like a universal Turing machine, but not of the power of Turing machines. The main problem is to encode the Petri nets to be simulated such that the universal net can also be encoded in that way.

In this paper we present some basic considerations of a general and uniform representation of such problems in terms of more general multisets. For that purpose we introduce an inductive definition of structured multisets (multisets of multisets, etc.) representing *data* as cell contents or molecules. Chomsky-like productions (rewriting rules) are also formulated as such higher order multisets. The productions are applied sequentially. Also regulated application as in matrix grammars is possible. Another possibility are parallel productions of a 0L type. The entire rewriting system, data and productions, can also be represented by a multiset (of still higher order). At that level it is possibe in principle to modify those productions by external ones (from a higher level). A main problem is to develop a simple and uniform definition how productions (of higher order) should deal with data (of lower order). Several possibilities for the order of application of productions are conceivable, from the inside to the outside, or from the outside to the inside, in particular moving through one membrane only.

Another goal is to construct a universal multiset rewriting system for simulating some (not necessarily recursively enumerable) class of multiset rewriting systems. The problem is to encode in some way multiset grammars. This can possibly results in considering other operations on multisets (different from + but still commutative and associative), and in a hierarchical structure similar to Scott's *domains*.

Thus, there are a number of open problems, the most important of them being the structure of multisets together with their operations to construct a universal multiset rewriting system below the power of Turing machines. This should be as simple as possible. The solution would allow us to construct corresponding universal Petri nets or P systems.

## 2 Definitions

Let $\Sigma = \{s_1, \cdots, s_k\}$ be a finite set of *atoms* (an *alphabet*). Let $\Sigma$ be ordered by $s_1 \sqsubseteq s_2 \sqsubseteq \cdots \sqsubseteq s_k$. A *multiset* on $\Sigma$ will be denoted by $\langle i_1 a_1, \cdots, i_r a_r \rangle$ with $1 \leq r \leq k$, $i_j > 0$, $a_j \in \Sigma$, and $i \neq j \Rightarrow a_i \neq a_j$.

Define inductively:

$$\begin{aligned}
&\mathcal{M}_0 = \Sigma,\\
&\mathcal{M}_1 = \{\langle i_1 a_1, \cdots, i_r a_r \rangle \mid 0 \leq r \leq k, i_j \in I\!N \setminus \{0\}, a_j \in \mathcal{M}_0\},\\
&\mathcal{M}_{s+1} = \{\langle i_1 m_1, \cdots, i_r m_r \rangle \mid r \in I\!N, i_j \in I\!N \setminus \{0\}, m_j \in \mathcal{M}_s\}.
\end{aligned}$$

Note that for $r = 0$ this defines the neutral element $\mathbf{0}_{s+1}$ for $\mathcal{M}_{s+1}$.

The elements of $\mathcal{M}_{s+1}$ can also be written as (formal) infinite sequences $\langle i_j \mid i_j \in I\!N \rangle$ where only finitely many $i_j \neq 0$, and the elements $m_j \in \mathcal{M}_s$ are ordered in some canonical way, e.g., lexicographically.

**Examples:** $\Sigma = \{a, b, c\}$,
$s = 0 : a, b, c$,
$s = 1 : \langle a, 2b\rangle, \langle c\rangle, \mathbf{0}_1$,
$s = 2 : \langle\langle a\rangle, 3\langle 2b, c\rangle\rangle, \langle\langle b\rangle\rangle, \langle\mathbf{0}_1\rangle$.

An order $\sqsubseteq_s$ on $\mathcal{M}_s$ $(s > 0)$ is defined by

$$m \sqsubseteq_s m' \Leftrightarrow \langle i_j\rangle \sqsubseteq_s \langle i'_j\rangle \Leftrightarrow \forall j : i_j \leq i'_j.$$

$\mathcal{M}_1$ is identical to the classical set of multisets: $\mathcal{M}_1 = I\!N^k$. It can describe molecules composed of atoms from $\Sigma$. $\mathcal{M}_2$ can describe sets of molecules.

Elements from $\mathcal{M}_s$ can also be interpreted as *cells* with a membrane denoted by $\langle$ and $\rangle$, containing cells of lower complexity. Note that $\mathcal{M}_s$ contains only multisets of the same kind.

With the operation $\oplus_s$ $(s > 0)$ defined by

$$m \oplus_s m' = \langle i_j\rangle \oplus_s \langle i'_j\rangle = \langle i_j + i'_j\rangle \text{ for } m, m' \in \mathcal{M}_s,$$

$\mathcal{M}_s$ becomes a commutative monoid with neutral element $\mathbf{0}_s$.

Therefore, the structure $(2^{\mathcal{M}_s}, \emptyset, \{\mathbf{0}_s\}, \cup, \oplus_s)$ is an $\omega$-complete semiring. This allows us to define *rational, linear,* and *algebraic* languages of multisets by least fixed points of corresponding systems of equations. Since the operation $\oplus_s$ is commutative, the families of rational, linear, and algebraic multiset languages coincide.

Another operation $\ominus_s$ is defined for $m \sqsubseteq_s m' \Leftrightarrow \langle i_j\rangle \sqsubseteq_s \langle i'_j\rangle$ by

$$m' \ominus_s m = \langle i'_j - i_j\rangle.$$

**Examples:** $\Sigma = \{a, b, c\}$,
$s = 1 : \langle a, 2b\rangle \oplus_1 \langle b, 2c\rangle = \langle a, 3b, 2c\rangle$,
$s = 2 : \langle\langle a\rangle, \langle a, 2b\rangle\rangle \oplus_2 \langle\langle a\rangle, \langle b\rangle\rangle = \langle 2\langle a\rangle, \langle b\rangle, \langle a, 2b\rangle\rangle$.

Another class of sets is defined inductively by:

$\mathcal{N}_0 = \mathcal{M}_0 = \Sigma \cup \{\mathbf{0}\}$,
$\mathcal{N}_1 = \mathcal{N}_0 \cup \{\langle i_1 a_1, \cdots, i_r a_r\rangle \mid 0 < r \leq k, i_j \in I\!N \setminus \{0\}, a_j \in \mathcal{N}_0 \setminus \{\mathbf{0}\}\}$,
$\mathcal{N}_{s+1} = \mathcal{N}_s \cup \{\langle i_1 m_1, \cdots, i_r m_r\rangle \mid r \in I\!N, i_j \in I\!N \setminus \{0\}, m_j \in \mathcal{N}_s \setminus \{\mathbf{0}\}\}$.

$\mathcal{N}_s$ contains multisets of different levels up to level $s$.

**Examples:** $\Sigma = \{a, b, c\}$,
$s = 0 : a, c$,
$s = 1 : b, \langle a\rangle, \langle 2a, b\rangle, \mathbf{0}$,
$s = 2 : a, \langle a, 3b\rangle, \langle a, \langle b\rangle, \langle 2a, 3b\rangle\rangle, \langle\langle 2a, 5c\rangle\rangle$.

Again, the elements of $\mathcal{N}_s$ can be ordered in some canonical way. Therefore, they can also be expressed as (formally) infinite sequences of multiplicities of elements from $\mathcal{N}_s : \langle i_j \mid i_j \in I\!N\rangle$ with only finitely many $i_j \neq 0$.

Elements from $\mathcal{N}_s$ can also be interpreted as *cells* with a membrane denoted by $\langle$ and $\rangle$, containing cells of possibly different lower complexities.

Again, an order $\sqsubseteq_s$ on $\mathcal{M}_s$ $(s > 0)$ can be defined by $\mathbf{0} \sqsubseteq m$ and

$$m \sqsubseteq_s m' \Leftrightarrow \langle i_j \rangle \sqsubseteq \langle i'_j \rangle \Leftrightarrow \forall j : i_j \leq i'_j.$$

Any element from $\mathcal{N}_s$ can be represented by a (formally) infinite sequence $\langle i_j \rangle$.

An associative operation $\otimes_s$ on $\mathcal{N}_s$ $(s > 0)$ can be defined by:

$\mathbf{0} \otimes_s m = m$ for $m \in \mathcal{N}_s$,
$a \otimes_s b = \langle a, b \rangle$ for $a, b \in \Sigma$,
$a \otimes_s \langle m_1, \cdots, m_r \rangle = \langle a, m_1, \cdots, m_r \rangle$, for $a \in \Sigma, \langle m_1, \cdots, m_r \rangle \in \mathcal{N}_s \setminus \Sigma$,
$m \otimes_s m' = \langle i_j \rangle \otimes_s \langle i'_j \rangle = \langle i_j + i'_j \rangle$, for $m, m' \in \mathcal{N}_s \setminus \Sigma$.

$\mathcal{N}_s$ thus becomes a commutative monoid with neutral element $\mathbf{0}$.

Another operation $\odot_s$ can be defined by

$$m' \odot_s m = \langle i'_j \rangle \odot_s \langle i_j \rangle = \langle i'_j - i_j \rangle \text{ for } m \sqsubseteq m'.$$

$\odot$ is defined in an analogous way.

Let $\mathcal{N}_* = \bigcup_{s \geq 0} \mathcal{N}_s$.

It is possible to represent $\Sigma^*$ in $\mathcal{N}_*$ inductively by $\mathbf{0}$ for $\lambda$, and $\langle a, \langle \beta \rangle \rangle$ for $a \in \Sigma$, $\beta \in \Sigma^*$.

Also on $\mathcal{N}_*$ a commutative operation $\otimes$ can be defined by

$$m \otimes m' = m \otimes_s m' \text{ if } m \in \mathcal{N}_t, m' \in \mathcal{N}_{t'} \text{ and } s = \max(t, t').$$

The structures$(2^{\mathcal{N}_s}, \emptyset, \{\imath\}, \cup, \otimes_s)$, $(2^{\mathcal{N}_*}, \emptyset, \{\imath\}, \cup, \otimes)$ are all $\omega$-complete semirings.

Therefore, *rational, linear,* and *algebraic* multiset languages of such structures can be defined as least fixed points of corresponding systems of equations. Again, since the operations are commutative, the rational, linear, and algebraic families of multiset languages coincide.

Since all systems of equations are finite, all multiset languages defined in this way contain only finitely many different 'symbols', in the case of $s = 2$, e.g., only finitely many of the form $\langle a, nb \rangle$ with $n > 0$. Therefore, to obtain more general classes of multiset languages, other methods to generate such multiset languages have to be investigated.

Another possibility to characterize multiset languages is to define grammars or rewriting systems on such structures.

For $\mathcal{M}_s$ the productions have the form $[\alpha, \beta]$ with $\alpha, \beta \in \mathcal{M}_s$. A simple rewriting step is given by the application of a production on $m \in \mathcal{M}_s$ yielding $m' \in \mathcal{M}_s$, and is defined by:

if $\alpha \sqsubseteq_s m$, then $m' = (m \ominus_s \alpha) \oplus_s \beta$,
or, for $\mathcal{N}_s$, by $m' = (m \odot_s \alpha) \otimes_s \beta$.

Again, such productions are too simple to generate multiset languages of higher complexity.

Consider multisets of level $s = 1$ as data, such as $\langle 3a, 2b\rangle$. Productions rewriting such multisets can be represented as multisets of higher level such as $\pi = \langle\langle A, \alpha\rangle, \langle C, \beta\rangle\rangle$ where $\alpha, \beta$ are multisets of level $s = 1$, and $A, C$ denote the *antecedent* and the *consequence* of a production, with $A, C \notin \Sigma$. $A, C$ are necessary to fix the order in $\pi$. Thus, a production $\pi$ is a multiset of level $s = 3$. A finite set of such productions then is a multiset $\Pi$ of level $s = 4$. Note that multiplicities at that level are 1. Finally, an entire rewriting system can be seen as a multiset of level $s = 5$. Productions have to be applied in the way as shown above.

This can be generalized for data of arbitrary level $s$. A rewriting system then can be interpreted as a multiset of level $s + 4$. Note that the level of data can be changed in this case.

**Examples:**

1. $\Sigma = \{a, b, c\}$,
   $\pi = \langle\langle A, \langle a, 2\langle b, 3c\rangle\rangle\rangle, \langle C, \langle 2b, c\rangle\rangle\rangle$,
   rewriting, e.g., $\langle 2a, 2\langle b, 3c\rangle\rangle$ to $\langle a, 2b, c\rangle$,
2. $\pi = \langle\langle A, \langle a\rangle\rangle, \langle C, \langle 2b, \langle a, c\rangle\rangle\rangle\rangle$,
   rewriting, e.g., $\langle a, c\rangle$ to $\langle 2b, c, \langle a, c\rangle\rangle$,
3. $\Sigma = \{h, o\}$,
   $\pi = \langle\langle A, \langle\langle 2\langle 2h\rangle, \langle 2o\rangle\rangle\rangle\rangle, \langle C, \langle 2\langle 2h, o\rangle\rangle\rangle\rangle$,
   describing the chemical reaction $2H_2 + O_2 \to 2H_2O$, or
4. $\pi = \langle\langle A, \langle\langle \alpha, \beta\rangle, \langle \gamma, \delta\rangle\rangle\rangle, \langle C, \langle\langle \alpha, \gamma\rangle, \langle \beta, \delta\rangle\rangle\rangle\rangle$,
   describing the reaction $\alpha\beta + \gamma\delta \to \alpha\gamma + \beta\delta$.

For $s = 1$ we just get the families of multiset languages **mARB** considered in [3].

In the case of (object) Petri nets as tokens of a (system) Petri net we have, e.g., a transition (production) of the system net like

$$\langle\langle\Pi\rangle, \alpha\rangle, 2\langle\langle\Pi\rangle, \beta\rangle\rangle \Rightarrow \langle 3\langle\langle\Pi\rangle, \gamma\rangle, \langle\langle\Pi\rangle, \delta\rangle\rangle,$$

where $\Pi$ describes the transitions (productions) of the object Petri net, and $\alpha, \beta, \gamma, \delta$ describe its configurations. Relations between the configurations are not given, but they should somehow reflect the behaviour of the object Petri net. This requires that information from productions have to be moved between different levels in some way.

In a more general situation also different object Petri nets (i.e. different $\Pi$) might be allowed.

As a simple example we give two representations of info-energy systems considered in [2]. The purpose is that information handling comsumes and creates (or transmits) energy.

Let $\mathbf{I}$ be a (finite) set of *informations*. To each $A \in \mathbf{I}$ an *energy* $k \in I\!N$ is attached. This is represented as a multiset $\langle kA \rangle$ for $k > 0$, and as a multiset $\langle A_0 \rangle$ for $k = 0$ with $\mathbf{I_0} = \{A_0 \mid A \in \mathbf{I}\}$.

A rule of such an information-energy system, denoting an energy flow of 3, is e.g.:

$\Sigma = \{A, B, A_0, B_0\}$,
$\langle\langle kA\rangle, \langle mB\rangle\rangle \to \langle\langle (k-3)A\rangle, \langle (m+3)B\rangle\rangle$ $(k > 3)$,
$\langle\langle 3A\rangle, \langle mB\rangle\rangle \to \langle\langle A_0\rangle, \langle (m+3)B\rangle\rangle$,
$\langle\langle kA\rangle, \langle B_0\rangle\rangle \to \langle\langle (k-3)A\rangle, \langle 3B\rangle\rangle$ $(k > 3)$,
$\langle\langle 3A\rangle, \langle B_0\rangle\rangle \to \langle\langle A_0\rangle, \langle 3B\rangle\rangle$.

Note that in these rules there is a relation (via $k \in I\!N$) between higher and lower level.

By these rules, a configuration

$$\langle\langle A_0\rangle, \langle 3A\rangle, 2\langle 5A\rangle, \langle B_0\rangle, \langle 4B\rangle\rangle$$

is transformed into

$$\langle 2\langle A_0\rangle, 2\langle 5A\rangle, \langle B_0\rangle, \langle 7B\rangle\rangle \quad \text{or} \quad \langle\langle A_0\rangle, \langle 2A\rangle, \langle 5A\rangle, \langle 3B\rangle, \langle 4B\rangle\rangle.$$

An alternative representation of information-energy is $\langle ke, A\rangle$ with $k \in I\!N$, and $\langle 0e, A\rangle$ meaning $\langle A\rangle$. The rule from above then may be written as

$$\langle\langle ke, A\rangle, \langle me, B\rangle\rangle \to \langle\langle (k-3)e, A\rangle, \langle (m+3)e, B\rangle\rangle \ (k \geq 3).$$

Since the order $\sqsubseteq_s$ on $\mathcal{M}_s$ is too restrictive, we define another order $\preceq_s$ on $\mathcal{M}_s$ for $s > 1$.

For $\alpha = \langle \alpha_1, \cdots, \alpha_k\rangle$, $\beta = \langle \beta_1, \cdots, \beta_l\rangle \in \mathcal{M}_{s+1}$ (all multiplicities written explicitely, i.e., some of the $\alpha_i, \beta_j \in \mathcal{M}_s$ may be identical) let

$$\alpha \preceq_{s+1} \beta \Leftrightarrow k \leq l \wedge \ \exists\, i_1, \cdots, i_k \ \forall j \in \{1, \cdots, k\} : \ \alpha_j \sqsubseteq \beta_{i_j}.$$

$\preceq_{s+1}$ is a reflexive, transitive, and antisymmetric relation. With this it is possible to define more general rewriting rules $\hat{\mu} \to \hat{\mu}'$ in the following way. If $\hat{\mu} = (\mu_1, \cdots, \mu_k)$, $\hat{\mu}' = (\mu'_1, \cdots, \mu'_k)$ with $\mu_i, \mu'_i \in \mathcal{M}_s$, let

$$\mu = \langle \mu_1, \cdots, \mu_k\rangle, \ \mu' = \langle \mu'_1, \cdots, \mu'_k\rangle \in \mathcal{M}_{s+1}.$$

If $\mu \preceq_{s+1} \alpha$, then $\mu_j \sqsubseteq_s \alpha_{i_j}$ for $j \in \{1, \cdots, k\}$. Note that the $i_j$ are not unique. A result of rewriting then is (among others) obtained by replacing the $\alpha_{i_j}$ in $\alpha$ by $(\alpha_{i_j} - \mu_j) + \mu'_j$.

In the example from above, with $(\langle 3e, A\rangle, \langle B\rangle) \to (\langle A\rangle, \langle 3e, B\rangle)$ the multiset $\langle\langle A\rangle, \langle 3e, A\rangle, \langle 5e, A\rangle, \langle 5e, A\rangle, \langle B\rangle, \langle 4e, B\rangle\rangle$ would be rewritten into the multiset $\langle\langle A\rangle, \langle 2e, A\rangle, \langle 3e, A\rangle, \langle 5e, A\rangle, \langle 3e, B\rangle, \langle 4e, B\rangle\rangle$ (among other possibilities).

However, if the resulting structure is meant not to have the power of Turing machines (there exist, e.g., universal DLBA's), the transmission of such information has to be chosen carefully. To see this, consider 2-counter machines which have of the power of Turing machines. They consist of finitely many

states, $Q = \{q_1, \cdots, q_m\}$ and two counters $c_1, c_2$ with transitions of the form $(q, x_1, x_2) \rightarrow (q', x_1', x_2')$ with $x_j' = x_j \pm 1$ and testing for $x_j = 0$ (in the case $x_j = 0$ only $x_j' = x_j + 1$ allowed). Note that a transition not changing a counter may be simulated by two transitions, the first one adding 1 to, and the second one subtracting 1 from that counter.

A configuration $(q, x_1, x_2)$ of a 2-counter machine can easily be represented by a multiset $\langle q, \langle c_1, x_1 a_1 \rangle, \langle c_2, x_2 b_2 \rangle \rangle$ or by $\langle q, x_1 a, x_2 b \rangle$.

If productions are ordered (have a priority, corresponding to appearance checking in matrix grammars) and erasing productions are allowed, then zero testing is possible already on the lowest level by

$\langle q, a \rangle \rightarrow \langle q' \rangle$, $\langle q \rangle \rightarrow \langle q', a \rangle$ if $(q, x) \rightarrow (q, x - 1)$, for $x > 0$ and $(q, 0) \rightarrow (q', 1)$, and

$\langle q, a \rangle \rightarrow \langle q', aa \rangle$, $\langle q \rangle \rightarrow \langle q', a \rangle$ if $(q, x) \rightarrow (q, x + 1)$, for $x > 0$ and $(q, 0) \rightarrow (q', 1)$.

This can easily be generalized to two counters giving a simple proof of the fact that multiset matrix grammars with erasing productions and appearance checking generate the recursively enumerable sets. Since only one membrane and one cell is necessary, this also shows that membrane computing is as powerful as Turing machines.

In this case we get 4 cases of transitions $(q, x_1, x_2) \rightarrow (q', x_1 \pm 1, x_2 \pm 1)$, denoted by $(++{+}{+}), (++{-}{+}), (-+{+}{+}), (-+{-}{+})$ (note that $x_i = 0$ only allows $x_i' = 1$):

$(++{+}{+}) : \langle q \rangle \rightarrow \langle q', a, b \rangle$,
$(++{-}{+}) : \langle q, b \rangle \rightarrow \langle q', a \rangle$, $\langle q \rangle \rightarrow \langle q', b \rangle$,
$(-+{+}{+}) : \langle q, a \rangle \rightarrow \langle q', b \rangle$, $\langle q \rangle \rightarrow \langle q', a \rangle$,
$(-+{-}{+}) : \langle q, a, b \rangle \rightarrow \langle q' \rangle$, $\langle q, a \rangle \rightarrow \langle q', b \rangle$, $\langle q, b \rangle \rightarrow \langle q', a \rangle$, $\langle q \rangle \rightarrow \langle q', a, b \rangle$.

## 3 Conclusion

The considerations outlined above are only a first step for the development of a simple uniform multiset model for describing multiset rewriting, some classes of Petri nets, and P systems, and to construct universal instances for them. A lot a further research has to be done in that direction.

**Acknowledgements**

We want to thank Berndt Farwer for many critical and useful remarks.

## References

1. R. Freund: *Sequential P-Systems.* Romanian Journal of Information Science and Technology **4** No. 1-2, pp. 77-88, 2001.
2. P. Frisco, S. Ji: *Towards a Hierarchy of Info-energy Systems.* In: Pre-proceedings of *WMC-CdeA2002*, eds. Gh. Păun, C. Zandron, 2002, pp. 249–263.
3. M. Kudlek, C. Martín Vide, Gh. Păun: *Toward FMT (Formal Macroset Theory).* In: *Multiset Processing*, eds. C. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, LNCS 2235, pp. 123-133, 2001.

4. Gh. Păun: *Computing with Membranes.* TUCS Research Report **208**, 1998 (`http//www.tucs.fi`).
5. Gh. Păun: *Computing with Membranes: An Introduction.* EATCS-Bulletin **67**, pp. 139-152, 1999.
6. Gh. Păun: *Computing with Membranes.* Journal of Computer and System Sciences **61.1**, pp. 108-143, 2000.

# A Survey of Some Variants of P Systems

Mutyam Madhu and Kamala Krithivasan

Theoretical Computer Science Laboratory
Dept. of Computer Science and Engineering
Indian Institute of Technology, Madras
Chennai-36, Tamil Nadu, India
madhu@cs.iitm.ernet.in
kamala@iitm.ernet.in

**Abstract.** In this paper we give a brief survey of some variants of P systems and their computational capacity. An improvement of a known result about rewriting P systems with leftmost derivation is given. We also give the generalized definition for normal forms of rewriting P systems.

## 1 Introduction

P systems are a class of distributed parallel computing devices of biochemical type, introduced in [14], which can be seen as a general computing architecture wherein various types of objects can be processed by various operations. In the basic model of the P systems one considers a *membrane structure* consisting of several cell-like membranes which are hierarchically embedded in a main membrane, called the *skin membrane*. Membranes with no other membranes embedded in them are called *elementary*. The membranes delimit *regions*, where we place *objects*. The objects evolve according to given *evolution rules* which are associated with the regions. An object can evolve independently of the other objects in the same region of the membrane structure, or in cooperation with other objects. In particular, we consider *catalysts*, objects which evolve only together with other objects, but are not modified by the evolution (they just help other objects to evolve). The evolution rules are given in the form of multiset transition rules, with an optional associated priority relation. The right hand side of the rules contains symbols of the form $(a, here), (a, out), (a, in_j)$, where $a$ is an object. The meaning is that one occurrence of the object $a$ is produced and remains in the same region, is sent out of the respective membrane, or is sent to membrane $j$ (which should be reachable from the region where the rule is applied), respectively. One can consider $(a, in)$ instead of $(a, in_j)$ with the meaning that one occurrence of the object $a$ is sent to any of the directly inner membranes.

A feature considered in [15] is the possibility to control the membrane thickness (by using the actions $\tau$ and $\delta$). This is done as follows: Initially, all membranes are considered to be of thickness 1. If a rule in a membrane of thickness 1 introduces the symbol $\tau$, then the thickness of the membrane increases to 2. A

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 360–370, 2003.

membrane of thickness 2 does not become thicker by using further rules which introduce the symbol $\tau$. If a rule which introduces the symbol $\delta$ is used in a membrane of thickness 1, then the membrane is dissolved; if the membrane had thickness 2, then it returns to thickness 1. Whenever the skin membrane is dissolved, the whole membrane system will be destroyed. If at the same step one uses rules which introduce both $\delta$ and $\tau$ in the same membrane, then the membrane does not change its thickness. No object can be communicated through a membrane of thickness 2, hence rules which introduce commands *in* or *out* requesting such communications, cannot be used. However, the communication has priority over changing the thickness: if at the same step an object should be communicated and a rule introduces the action $\tau$, then the object is communicated and after that the membrane changes the thickness.

The application of evolution rules is done in parallel. Starting from an initial configuration (identified by the membrane structure, the objects – with multiplicities – and rules placed in its regions) and using the evolution rules, we get a *computation*. We consider a computation complete when it halts, no further rule can be applied. Two ways of assigning a result to a computation are considered: (a) by designating an internal membrane as the output membrane, (b) by reading the result outside the system, where the output is obtained in the natural way: we arrange the symbols leaving the system, in the order they are expelled from the skin membrane; when several objects exit at the same time, any permutation of them is accepted.

## 2 P Systems with Active Membranes

P systems with active membranes were introduced in [17] and then investigated in [12], [16], and [2]. In this section, we consider the *restricted version of P systems with active membranes* as given in [12].

A *P system with active membranes, in the restricted form,* is a construct

$$\Pi = (V, T, H, \mu, M_1, \cdots, M_n, R),$$

where:

- $V$ is the alphabet of the system;
- $T \subseteq V$ is the terminal alphabet;
- $H$ is a finite set of *labels* for membranes;
- $\mu$ is a *membrane structure,* consisting of $n$ membranes labeled with elements of $H$ and initially having a neutral charge;
- $M_i, 1 \leq i \leq n$, are strings over $V$, describing the *multisets of objects* placed in the $n$ regions of $\mu$;
- $R$ is a finite set of *rules*, of the following forms:
  - **(a)** $[_h a \rightarrow v]_h^\alpha$, for $h \in H, a \in V, v \in V^*, \alpha \in \{+, -, 0\}$ (object evolution rules),
  - **(b)** $a[_h]_h^\alpha \rightarrow [_h b]_h^\beta$, where $a, b \in V, h \in H, \alpha, \beta \in \{+, -, 0\}$ (an object is introduced in membrane $h$),

**(c)** $[_h a]_h^\alpha \to [_h]_h^\beta b$, where $a, b \in V, h \in H, \alpha, \beta \in \{+, -, 0\}$ (an object is sent out of membrane $h$).

The rules are applied as usual in a P system, in a maximally parallel manner: in each time unit, all objects which can evolve, have to evolve. Each copy of an object and each copy of a membrane can be used by only one rule, with the exception of rules of types (a), where we count only the involved object, not the membrane. That is, if we have several objects $a$ in a membrane $i$ and a rule $[_i a \to v]_i^\alpha$, then we use this rule for all copies of $a$; we do not count the membrane that was used – note that its electrical charge is not changed. However, if we have a rule $[_i a]_i^\alpha \to [_i]_i^\beta b$, then this counts as using the membrane, no other rule of types (b) and (c) which uses the same membrane can be used at the same time.

As any other membrane, the skin membrane can be *electrically charged*. During a computation, objects can leave the skin membrane (using rules of type (c)).

The result of a successful computation is $\Psi_T(w)$, where $w$ describes the multiset of objects over $T$ which have left the skin membrane during the computation. The set of such vectors $\Psi_T(w)$ is denoted by $Ps(\Pi)$ and we say that it is *generated* by $\Pi$. The family of all such sets of vectors, computed by systems with at most $n, n \geq 1$, membranes, is denoted by $PsAP_n$; when the number of membranes is not restricted, we replace the subscript $n$ by $*$.

In [17], the universality of P systems with active membranes (for the general case) is proven. However, in [12] without considering membrane division, it is proved that $PsRE = PsAP_*$. For this restricted case (i.e., without membrane division) in [2] it is proved that four membranes suffice, i.e., $PsRE = PsAP_4$. The following result, for the restricted case, can be found in [10]:

**Theorem 1.** $PsRE = PsAP_3$.

## 3 P Systems with Membrane Creation

In [6], we considered a variant of P systems where each membrane has both *productive* and *non-productive* objects. A *productive* object can create a new membrane and transforms into other objects. A *non-productive* object only transforms into other objects without creating a new membrane.

Formally, a *P system with membrane creation* is a construct

$$\Pi = (V, T, C, \mu, M_1, \cdots, M_m, R_1, \cdots, R_n),$$

where $V, T, \mu, M_i, 1 \leq i \leq m$, are defined in the usual way and $C, C \cap V = \emptyset$, is a finite set of catalysts. Each $R_i, 1 \leq i \leq n$, is a finite set of evolution rules over $V \cup C$. An evolution rule can be of two types:

- $a \to v$ or $ca \to cv$, where $c \in C, a \in V, v = v'$ or $v = v'\delta$ or $v = v'\tau$, where $v'$ is a multiset of objects over $(V \times \{here, out, in\})$ and $\delta, \tau$ are special symbols not in $V$; the object $a$ is said to be *non-productive*;

– $a \to [_i\ v]_i$, where $v$ is a multiset of objects over $V$; this rule indicates that the object identified by $a$ is transformed into the objects identified by $v$, surrounded by a new membrane having the label $i$. No rule of the form $a \to [_1\ v]_1$ can appear in any set $R_i$, because a membrane with index 1 indicates the skin membrane, which cannot be duplicated. The object $a$ is said to be *productive*.

The rules are applied in the standard manner in P systems, that is, in the non-deterministic maximally parallel way, with the mentioning that using a rule $a \to [_i v]_i$ in a membrane $j$ means creating a new membrane, with the label $i$, inside membrane $j$, and containing the objects specified by $v$. Note that by knowing the label $i$ of the new membrane we know the rules which are to be applied to its objects, namely those from the set $R_i$.

The number of initial membranes ($n_1$), the maximal number of membranes simultaneously present in the system ($n_2$), and the number of all possible types of membranes ($n_3$) form the *profile* $(n_1, n_2, n_3)$ of the system.

The result of a successful computation is defined in the usual way. The family of all sets of vectors of natural numbers generated by P systems with membrane creation with a profile component-wise smaller than $(n_1, n_2, n_3)$, with catalysts and the actions of both $\delta, \tau$, using the target indications of the form $here, out, in$, is denoted by $PsPMC_{(n_1,n_2,n_3)}(Cat, i/o, \delta, \tau)$; when one of the features $\alpha \in \{Cat, \delta, \tau\}$ is not present, we replace it with $n\alpha$. When we use the communication commands of the form $here, out, in_j$, we replace $i/o$ with $tar$.

In [6], we proved that $PsRE$ is equal to $PsPMC_{(1,2,4)}(Cat, tar, \tau, \delta)$. By considering *catalyst creation rules* [18] of the form $X \to [yc]$, where an object $X$ creates a membrane along with a catalyst $c$ and transform into $y$, it is proved that $PsRE$ is equal to $PsPMC_{(1,2,4)}(Cat, i/o, n\tau, \delta)$. Therefore, in [18] it is allowed the catalyst to be generated. Using this feature, in [10] one proves the following result:

**Theorem 2.** $PsRE = PsPMC_{(1,2,3)}(Cat, i/o, n\tau, \delta)$.

## 4 P Systems with Leftmost Derivation

In a cell, many objects can be considered as being *atomic*, but many other objects, such as, e.g., DNA molecules, have a structure, which can be described by a string. This suggests to consider *P systems with string-objects* [11]. One natural way to process string-objects is to use rules of the form $X \to (v, tar)$, where $X \to v$ is a usual context-free rule and $tar$ is a target indication, one of *here, out, in,* specifying in the standard way the region where the result of rewriting should go. All strings are processed in parallel, but each single string is rewritten by only one rule (the parallelism is maximal at the level of strings and rules, but the rewriting is sequential at the level of the symbols from each string).

A restriction in the use of rules of *P systems with string-objects* is considered in *P systems with leftmost derivation* [1], where any string is rewritten in the

leftmost position which can be rewritten by a rule from its region. In order to apply a rule, we examine the symbols of the string, step by step, from left to right and the first symbol which can be rewritten by a rule from the region of the string is rewritten. If there are several rules with the same left-hand side symbol, then we can select one of them nondeterministically.

Formally, a *P system with leftmost derivation* is a construct

$$\Pi = (V, T, \mu, L_1, \cdots, L_n, R_1, \cdots, R_n),$$

where $V, T, \mu, R_i, 1 \leq i \leq n$, are defined in the usual way and each $L_i, 1 \leq i \leq n$, is a language over $V$, representing the strings initially present in the regions $1, 2, \cdots, n$ of $\mu$.

A computation is defined in the usual way. The result of a successful computation consists of the strings over $T$ ejected from the skin membrane. (Note that the strings which remain in the system, as well as the strings which exit the system but contain symbols which do not belong to $T$ are ignored.)

We denote by $RP_n(left), n \geq 1$, the family of languages generated by P systems with leftmost derivation with at most $n$ membranes.

From [1], we know that:

**Theorem 3.** $RE = RP_6(left)$.

We improve this result and show that universality can be achieved with *five* membranes.

**Theorem 4.** $RE = RP_5(left)$.

*Proof.* We prove only the inclusion $RE \subseteq RP_5(left)$. The inclusion in the other direction can be proved in a straightforward manner. Let us consider a matrix grammar with appearance checking, $G = (N, T, S, M, F)$, in the strong binary normal form [3] with $N = N_1 \cup N_2 \cup \{S, \dagger\}$ and $ac(G) \leq 2$. Assume that $ac(G) = 2$, and let $B^{(1)}$ and $B^{(2)}$ be the two symbols in $N_2$ for which we have rules $B^{(j)} \to \dagger, j \in \{1, 2\}$, in matrices of $M$. Let us assume that we have $h$ matrices of the form $m'_i : (X \to Y, B^{(j)} \to \dagger), X, Y \in N_1, j \in \{1, 2\}, 1 \leq i \leq h$, and $k$ matrices of the form $m_i : (X \to \alpha, A \to x), X \in N_1, A \in N_2, \alpha \in N_1 \cup \{\lambda\}$, and $x \in (N_2 \cup T)^*, 1 \leq i \leq k$. Each matrix of the form $(X \to \lambda, A \to x), X \in N_1, A \in N_2, x \in T^*$, is replaced by $(X \to f, A \to x)$, where $f$ is a new symbol. We continue to label the obtained matrix in the same way as the original one. The matrices of the form $(X \to Y, B^{(j)} \to \dagger), X, Y \in N_1$, are labeled by $m'_i$, with $i \in lab_j$, for $j \in \{1, 2\}$, such that $lab_1, lab_2$ and $lab_0 = \{1, 2, \cdots, k\}$ are mutually disjoint sets.

We construct a P system

$$\Pi = (V, T, \mu, L_1, \cdots, L_5, R_1, \cdots, R_5),$$

with the following components:

- $V = N_1 \cup N_2 \cup T \cup \{X_i \mid X \in N_1, i \in lab_0 \cup lab_1 \cup lab_2\}$
  $\cup \{A', A_i \mid A \in N_2, i \in lab_0\} \cup \{f, \dagger\}$;

- $\mu = [_1[_2[_3\ ]_3]_2[_4\ ]_4[_5\ ]_5]_1$;
- $L_1 = \{AX\}$;
- $L_i = \emptyset, 2 \leq i \leq 5$;
- $R_1$ contains the following rules:
  1. $X \rightarrow (Y_i, in), X \in N_1, i \in lab_0 \cup lab_1 \cup lab_2$;
  2. $A' \rightarrow (A, here), A \in N_2 - \{B^{(1)}, B^{(2)}\}$;
  3. $Y_i \rightarrow (\dagger, here), i \in lab_0 \cup lab_1 \cup lab_2$;
  4. $\dagger \rightarrow (\dagger, here)$;
  5. $f \rightarrow (\lambda, out)$;
- $R_2$ contains the following rules:
  1. $A \rightarrow (A', here), A \in N_2 - \{B^{(1)}, B^{(2)}\}$;
  2. $A \rightarrow (A_1, in), A \in N_2 - \{B^{(1)}, B^{(2)}\}$;
  3. $A_i \rightarrow (A_{i+1}, in), A \in N_2 - \{B^{(1)}, B^{(2)}\}, 1 \leq i \leq k-1$;
  4. $A_i \rightarrow (x, out)$, for $m_i : (X \rightarrow Y, A \rightarrow x), i \in lab_0$;
- $R_3$ contains the following rules:
  1. $Y_i \rightarrow (Y_{i-1}, out), i \neq 1$;
  2. $Y_1 \rightarrow (Y, out)$;
  3. $Y \rightarrow (\dagger, here)$;
  4. $\dagger \rightarrow (\dagger, here)$;
- $R_4$ contains the following rules:
  1. $B^{(1)} \rightarrow (\dagger, here)$;
  2. $\dagger \rightarrow (\dagger, here)$;
  3. $Y_i \rightarrow (Y, out), i \in lab_1$;
  4. $Y_i \rightarrow (\dagger, here), i \notin lab_1$;
- $R_5$ contains the following rules:
  1. $B^{(2)} \rightarrow (\dagger, here)$;
  2. $\dagger \rightarrow (\dagger, here)$;
  3. $Y_i \rightarrow (Y, out), i \in lab_2$;
  4. $Y_i \rightarrow (\dagger, here), i \notin lab_2$;

The system works as follows:

We start with the string $AX$ in membrane 1; assume that we have here a string $wX$, for some $w \in (N \cup T)^*$. Since only one rule (i.e., $X \rightarrow (Y_i, in)$) can be applied to this string in membrane 1, we rewrite $X$ with $Y_i, i \in lab_0 \cup lab_1 \cup lab_2$, so that the string is sent to one of the inner membranes. If $i \in lab_0$ and the string is sent to either membrane 4 or 5, then a trap symbol ($\dagger$) will be introduced and the computation never halts. Otherwise, if $i \in lab_0$ and the string is sent to membrane 2, then we can apply any of the two rules $A \rightarrow (A', here)$ and $A \rightarrow (A_1, in), A \in N_2 - \{B^{(1)}, B^{(2)}\}$. If we apply only the first rule to all possible symbols from set $N_2 - \{B^{(1)}, B^{(2)}\}$, then we cannot proceed further and never get the output. In order to proceed further we need to apply the second rule to at least one symbol from the set $N_2 - \{B^{(1)}, B^{(2)}\}$, so that the string is sent to membrane 3. In membrane 3, we apply the rule $Y_i \rightarrow (Y_{i-1}, out), i \neq 1$, so that the string is sent to membrane 2. Now in membrane 2, since we cannot rewrite symbols of the form $A', A \in N_2 - \{B^{(1)}, B^{(2)}\}$, we apply the rule $A_i \rightarrow (A_{i+1}, in), A \in N_2 - \{B^{(1)}, B^{(2)}\}, 1 \leq i \leq k-1$, which makes the

string to move into membrane 3. In this way we decrease the subscript value of $Y$ in one step of rewriting and in the next step we increase the subscript value of $A$. After rewriting $Y_1$ into $Y$ we can apply either $A_i \to (A_{i+1}, in), A \in N_2 - \{B^{(1)}, B^{(2)}\}, 1 \leq i \leq k-1$, or $A_i \to (x, out)$, for $m_i : (X \to Y, A \to x)$ (in membrane 2). If we apply the first rule, then a trap symbol will be introduced in membrane 3 (due to the rule $Y \to (\dagger, here)$). So, with the application of the second rule we can finish the simulation of a matrix $m_i, i \in lab_0$. If we simulated a matrix $m_i : (X \to f, A \to x), i \in lab_0$, then the symbol $f$ will be erased in the skin membrane and the string is sent out of the system.

For simulating a matrix of the form $(X \to Y, B^{(1)} \to \dagger)$, we rewrite $X$ into $Y_i, i \in lab_1$, so that the string is sent to one of the inner membranes. If this string is moved into either membrane 2 or 5, then a trap symbol will be introduced. Otherwise, in membrane 4, we can apply the rule $B^{(1)} \to (\dagger, here)$ (if such a symbol exists) so that the computation never halts. If no such symbol (i.e., $B^{(1)}$) exists, then we can apply the rule $Y_i \to (Y, out)$ and finish the simulation of a matrix $m'_i : (X \to Y, B^{(1)} \to \dagger)$. Similarly, we can explain the simulation of a matrix $m'_i : (X \to Y, B^{(2)} \to \dagger)$.

Therefore our system generates exactly the strings generated by the grammar $G$, that is, we have $L(\Pi) = L(G)$. □

## 5 Contextual P Systems

Contextual grammars for processing string-objects in P systems were considered in [7,5], where the derivations are taking place depending upon the contexts.

Formally, a *contextual P system* is a construct

$$\Pi = (V, T, \mu, L_1, \cdots, L_n, R_1, \cdots, R_n),$$

where $V, T, \mu, L_i, 1 \leq i \leq n$, are defined in the usual way and each $R_i, 1 \leq i \leq n$, is a finite set of rules of the form $(x, (u, v), tar)$, where $x, u, v \in V^*$, and $tar \in \{here, in, out\}$. The pair $(u, v)$ is called the context and the string $x$ is called the selector of the rule. If we do not have any selection of contexts (the contexts are freely adjoined, irrespective of the bracketed substring), then the rules are represented as $(-, (u, v), tar)$.

We allow empty contexts, i.e., of the form $(\lambda, \lambda)$. A computation is defined in the usual way. During a computation, depending upon the selector, a context is attached to a string and then we follow the prescriptions given by $tar$. The result of a successful computation consists of the strings over $T$ ejected from the skin membrane.

We denote by $ICC$ and $ECC$ the family of languages generated by internal and external contextual grammars, respectively. If we do not consider a selection, then the corresponding families of languages are denoted by $IC$ and $EC$, respectively. The family of languages generated by total contextual grammars is denoted by $TC$. (Definitions of all these families can be found in [13].)

The family of languages generated by contextual P systems of degree $n, n \geq 1$, in the mode $X \in \{IC, ICC, EC, ECC\}$ with the selectors of type $F \in$

$\{FIN, REG, LIN, CF, CS, RE\}$, is denoted by $CP_n(X, F, i/o)$. Note that for $X \in \{IC, EC\}$, $F$ is $\emptyset$.

From [7], we know the following results:

**Theorem 5.** *1.* $CP_1(EC, \emptyset, i/o) - ECC \neq \emptyset$.
*2.* $CP_2(ECC, LIN, i/o) - TC \neq \emptyset$.
*3.* $CP_2(ECC, LIN, i/o) - CF \neq \emptyset$.
*4.* $X(F) \subseteq CP_1(X, F, i/o)$,
$X \in \{IC, ICC, EC, ECC\}, F \in \{FIN, REG, LIN, CF, CS, RE\}$.

When considering rules of the form $(x, (u, \lambda), tar)$ or $(x, (\lambda, v), tar)$, where $x, u, v \in V^*$ and $tar \in \{here, in, out\}$, we obtain *one-sided contextual P systems.*

The family of languages generated by one-sided contextual P systems of degree $n, n \geq 1$, in the mode $X \in \{IC, ICC, EC, ECC\}$ with the choice of type $F \in \{FIN, REG, LIN, CF, CS, RE\}$, is denoted by $OCP_n(1X, F, i/o)$ or $OCP_n(11X, F, i/o)$, depending upon the use of right-sided contexts or both right-sided and left-sided contexts, respectively.

From [7], we know that:

**Theorem 6.** *1.* $OCP_2(11EC, \emptyset, i/o) - 11TC \neq \emptyset$.
*2.* $OCP_2(1IC, \emptyset, i/o) - 11TC \neq \emptyset$.
*3.* $OCP_2(11EC, \emptyset, i/o) - REG \neq \emptyset$.

In an *insertion contextual P system*, we consider a finite set of rules of the form $((u, x, v), tar)$, where $x, u, v \in V^*$ and $tar \in \{here, in, out\}$. The meaning of a triple $(u, x, v)$ is: a string $x$ can be inserted in the context pair $(u, v)$. The weight of the system is defined as the maximum length of contexts used in all rules.

The family of languages generated by insertion contextual P systems of degree $n, n \geq 1$, with weight $m, m \geq 1$, is denoted by $ICP_n(m, i/o)$.

Let the family of languages generated by insertion grammars [13] of weight at most $n$, $n \geq 0$, is denoted by $S_n$; the union of all these families is denoted by $S_\infty$. Then, from [7], we have the following results:

**Theorem 7.** *1.* $ICP_2(1, i/o) - S_\infty \neq \emptyset$.
*2.* $ICP_2(1, i/o) - CF \neq \emptyset$.

So far in all variants of contextual P systems, we consider only contexts to be attached. By considering erased contexts, along with attached contexts, we obtain a *contextual P system with erased contexts.*

The family of languages generated by contextual P systems with erased contexts of degree $n, n \geq 1$, in the mode $X \in \{IC, ICC, EC, ECC\}$ with the choice of type $F \in \{FIN, REG, LIN, CF, CS, RE\}$, is denoted by $CPE_n(X_d, F, i/o)$.

From [7], we know that:

**Theorem 8.** *1.* $X_d(F) \subseteq CPE_1(X_d, F, i/o), X \in \{ICC, ECC\}, F \in \{FIN, REG, LIN, CF, CS, RE\}$.
*2.* $RE = CPE_1(ICC_d, FIN, i/o)$.

As a restricted case to erased contexts we define *one-sided erased contexts*, where we can erase a context either from right-side or from left-side but not from both sides at a time. We denote by $1X_d$ the family of languages generated by type $X$ grammars with right-sided contexts $(\lambda, v)$, and by $11X$ the family of languages generated when both right-sided contexts $(\lambda, v)$ and left-sided contexts $(u, \lambda)$ are allowed (but not contexts $(u, v)$ with both $u$ and $v$ non-empty), $X \in \{EC, ECC, IC, ICC\}$.

A *one-sided contextual P system with one-sided erased contexts* is a system where attached and erased contexts are one-sided.

Let $OCPOE_n(1Z, F_1, 1X_d, F_2, i/o)$ denote the family of languages generated by right-sided contextual P systems of degree $n$ in $Z$ mode with $F_1$ choice and with right-sided erased contexts in $X$ mode with $F_2$ choice. Here $X, Z \in \{IC, ICC, EC, ECC\}$ and $F_1, F_2 \in \{FIN, REG, LIN, CF, CS, RE\}$. If we use both right-sided and left-sided contexts, then the corresponding family of languages is denoted by $OCPOE_n(11Z, F_1, 11X_d, F_2, i/o)$.

The following theorem can be found in [7]:

**Theorem 9.** $RE = OCPOE_2(1ICC, FIN, 1IC_d, \emptyset, i/o)$.

By combining insertion rules with one-sided erased context rules, we obtain *insertion contextual P systems with one-sided erased contexts.*

Let $ICPOE_n(k, 1X_d, F, i/o)$ denote the family of languages generated by insertion contextual P systems of degree at most $n$ with weight at most $k$ and with right-sided erased contexts of mode $X$ with $F$ choice. Here $X \in \{IC, ICC, EC, ECC\}$ and $F \in \{FIN, REG, LIN, CF, CS, RE\}$. If we use both right-sided and left-sided erased contexts, then the corresponding family of languages is denoted by $ICPOE_n(k, 11X_d, F, i/o)$.

The proof of the following theorem can be found in [7].

**Theorem 10.** $RE = ICPOE_2(1, 1IC_d, \emptyset, i/o)$.

## 6 Hybrid P Systems

A hybrid P system is a construct

$$\Pi = (V, T, \mu, L_1, \cdots, L_n, R_1, \cdots, R_n, k),$$

where $V, T, \mu, L_i, 1 \leq i \leq n$, are defined in the usual way and each $R_i, 1 \leq i \leq n$, contains both rewriting and contextual rules; $k, 1 \leq k \leq n$, is an output membrane.

The family of languages generated by hybrid P systems of degree $n, n \geq 1$, with the choice $X \in \{FIN, REG, CF, CS, RE\}$ and $tar \in \{here, out, in_j\}$, is denoted as $HyP_n(X, tar)$. If $tar \in \{here, out, in\}$, then the family of languages generated is denoted by $HyP_n(X, i/o)$.

In [4], it is shown that *seven* membranes are needed for the universality of hybrid P systems with *regular* choice. In [9], we improved this result and showed that *two* membranes suffice for achieving the universality of hybrid P systems

with *regular* choice. Similarly, with *finite* choice, we achieved the universality of hybrid P systems with *five* membranes.

**Theorem 11.** *1.* $RE = HyP_2(REG, i/o)$.
*2.* $RE = HyP_5(FIN, tar)$.

## 7 Generalized Normal Form for Rewriting P Systems

A rewriting P system is in the $m\_n\_normal\_form$ [8] if it is of depth $m$ and in each membrane we have exactly $n$ rewriting rules. If we put no restriction either on the depth or on the number of rewriting rules, then we replace the corresponding term (i.e., either $m$ or $n$ respectively) with $*$.

The following results can be found in [8]:

**Theorem 12.** *1. Every recursively enumerable language can be generated by a rewriting system with priorities in* $2\_2\_normal\_form$.
*2. Every language generated by a rewriting P system of degree m in* $k\_*\_normal\_form$ *with/without priorities can be generated by a rewriting P system of degree m in* $2\_*\_normal\_form$ *with priorities.*

## References


1. C. Ferretti, G. Mauri, Gh. Păun, C. Zandron, On three variants of P systems with string-objects, in *Pre-proceedings of Workshop on Membrane Computing*, Curtea de Argeş, Romania, August 2001, Technical Report 17/01 of Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain, 2001, 63–76.
2. R. Freund, C. Martin-Vide, Gh. Păun, Computing with membranes. Three more collapsing hierarchies, submitted, 2000.
3. R. Freund, Gh. Păun, On the number of non-terminals in graph-controlled, programmed, and matrix grammars, in *Proc. of Third. Intern. Conf. on Machines, Computation, and Universality*, Chişinău, Moldova, 2001 (M. Margenstern, Y. Rogozhin, eds.), *LNCS* 2055, Springer-Verlag, 2001, 214–225.
4. S.N. Krishna, K. Lakshmanan, R. Rama, Hybrid P systems, *Romanian J. Information Science and Technology*, 4, 1 (2001), 111–124.
5. S.N. Krishna, K. Lakshmanan, R. Rama, On the power of P systems with contextual rules, *Fundamenta Informaticae*, 49, 1-3 (2002), 167–178.
6. M. Madhu, K. Krithivasan, P systems with membrane creation: Universality and efficiency, in *Proc. of Third. Intern. Conf. on Machines, Computation, and Universality*, Chişinău, Moldova, 2001 (M. Margenstern, Y. Rogozhin, eds.), *LNCS* 2055, Springer-Verlag, 2001, 276–287.
7. M. Madhu, K. Krithivasan, Contextual P systems, *Fundamenta Informaticae*, 49, 1-3 (2002), 179–189.
8. M. Madhu, K. Krithivasan, Generalized normal forms for rewriting P systems, *Acta Informatica*, 38, 10 (2002), 721–734.
9. M. Madhu, K. Krithivasan, Hybrid P systems: Improved universality results, *Third Intern. Conf. on Unconventional Models of Computation*, Himeji, Japan, 2002.

10. M. Madhu, K. Krithivasan, Improved results about universality of P systems, *Bulletin of the EATCS*, 76 (2002), 162–168.
11. C. Martin-Vide, Gh. Păun, String objects in P systems, in *Proc. of Algebraic Systems, Formal Languages, and Computation Workshop*, RIMS Kokyuroku, Kyoto Univ., 2000, 161–169.
12. A. Păun, On P systems with membrane division, in *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer-Verlag, 2000, 187–201.
13. Gh. Păun, *Marcus Contextual Grammars*, Kluwer Academic Publ., 1997.
14. Gh. Păun, Computing with membranes, *J. Computer System Sciences*, 61 (2000), 108–143.
15. Gh. Păun, Computing with membranes – A variant: P systems with polarized membranes, *Intern. J. Foundations of Computer Science*, 11 (2000), 167–182.
16. Gh. Păun, Computing with membranes; Attaching NP-complete problems, in *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer-Verlag, 2000, 94–115.
17. Gh. Păun, P systems with active membranes: Attacking NP-complete problems, *J. Automata, Languages and Combinatorics*, 11 (2001), 75–90.
18. R. Rodriguez-Paton, On the universality of P systems with membrane creation, *Bulletin of the EATCS*, 74 (2001), 229–233.

# Bridging P Systems and Genomics: A Preliminary Approach

Solomon Marcus

Romanian Academy, Mathematics
Calea Victoriei 125, Bucureşti, Romania
solomon.marcus@imar.ro

**Abstract.** Bringing genomics within the framework of P systems could give to the former the possibility to take profit of the computational capacities of the latter. Moreover, suggestions coming from genomics could enrich the study of P systems with new biological and computational ideas. In what follows, a first attempt is made in this respect.

## 1 "Life Is a Surface Activity"

"Life is a surface activity[...]. Life is fundamentally about insides and outsides" [5: 260]. Relevant parts of the environment are internalised as an "inside exterior" or "inner outside" (the so-called Uexküll's *Umwelt* [8]; "the representation of certain environmental features inside an organism by various means" [8: 28]), while the interior becomes externalised as an "outside interior" or "outer inside", in the form of the "semiotic niche" ([4: 40]), as informed and changed by the inside needs of the organism pertaining to that niche [3: 29]. This inside-outside interplay is made possible by the membrane strictly governing the traffic between them. P systems [7] find their starting point in this biological reality, to which a computational dimension is added. In agreement with the ideas of DNA computing and membrane computing, Wolfram [9] proposed recently to see life as a universal Turing machine, to which Chaitin [2] adds the condition of a high program-size complexity. The project of bridging genomics and P systems could have the slogan: *Life is DNA software + membrane software.*

## 2 P Systems and the Human Genome Project (HGP)

The HGP, as presented, in its computational aspect, by Karp [6], is a good starting point for the problem raised in the title of this article.

Let us recall the notion of a P system in one of its standard representations [1: 18]. A *P system with replicated rewriting* is a construct

$$\Pi = (V, T, \mu, M_1, M_2, \ldots, M_m, R_1, R_2, \ldots, R_m),$$

where $V$ is an *alphabet* (its elements are called *objects*); $T$ is contained in $V$ and it is called the *output alphabet*; $\mu$ is a *membrane structure* consisting of $m$

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 371–376, 2003.

membranes (or regions of a membrane) labeled by $1, 2, \ldots, m$, such that each membrane except the first is completely contained within another; $M_1, \ldots, M_m$ are finite languages over $V$; $R_1, \ldots, R_m$ are finite sets of *developmental rules* of the form $a \to (v_1, tar_1)|| \ldots ||(v_n, tar_n)$, with $n \geq 1$, where $tar_i$ belongs to $\{here, out, in\}$, $a \in V$, and $v_i \in V^*, \leq i \leq n$. The languages $M_i$ and the rule sets $R_i$ are associated with the regions of $\mu$, for all $1 \leq i \leq m$. When a rule of the form above is used to rewrite a string of the form $xay$, $n$ strings $xv_iy$ are obtained and sent to the regions indicated by $tar_i$. When $tar_i = here$, the string is kept in the same region. When $tar_i = out$, the string leaves the current region and goes into the outer one, which for region 1 means out of the system. When $tar_i = in$, the string is sent to one of the directly included regions, if any exists, otherwise the rule cannot be applied. A computation is defined as follows: the process starts with the strings present in the initial configuration and proceeds iteratively by applying in parallel the rules in each region to all strings that can be rewritten. If more than one rule can be applied to the same string in the same region, then only one, randomly chosen, rule will be applied. If the chosen rule can be applied in several places of the string, then it is applied in only one, randomly chosen, place. The result, the set of all terminal strings, is collected outside the system, at the end of the halting computation. The language generated by a system $\Pi$ is denoted by $L(\Pi)$ and consists of all strings over $T$ that are sent out of the system during a halting computation.

Our option for the variant investigated in [1] is motivated mainly by the fact that it distinguishes between the input and the output alphabet.

## 3 From Genomics to P Systems and Back

Let us recall that the genome of an organism is its total content of DNA molecules within the cromosomes. Each species has its genome characteristics and each individual within a species has its specific features. The human genome includes about three billion base pairs and about 35,000 genes. Our aim in the following is to identify the ways genomics, i.e., the study of genome, may lead naturally to some P systems. To the extent to which this task is fulfilled, the computational aspects of genomics may take profit from the computational capacities of P systems.

The usual, starting interpretation of the objects forming the alphabet of a P system is to consider them as molecules. The general theory of P systems does not depend on the way we interpret these objects; however, the intuitive representation of them decides to a large extent the type of problems which are investigated.

According to Karp [6], the main problems of genomics are: (a) to sequence and compare the genomes of different species (to sequence a DNA means to decompose it in its successive nucleotide bases); the sequencing of the human genome began in 1990 and was essentially completed in February 2001; (b) to identify the genes and determine the functions of the proteins they encode. Task (a) is mainly of a syntactic nature, while task (b) refers to the semantic dimension

of cellular processes. Some other tasks of the HGP were considered, but we leave them aside now.

A natural question arises: Which are (if they exist) the P systems accounting for the above tasks (a) and (b)? Referring to P systems with replicated rewriting, a first idea is to work with an alphabet $V$ including both the types of nucleotide bases and the types of amino acids, while the output alphabet $T$ contained in $V$ will be the set of various types of amino acids. The P system we are looking for should describe the process leading from DNA to its segmentation in nucleotide bases, from this segmentation to the identification of genes, which are privileged substrings of DNA, carrying the genetic information, and finally from genes to protein functions (the latter being hypothetically related to the protein sequencing, i.e., to their decomposition in amino acids). So, the membrane structure should consists of several regions, such as: a region of nucleotide bases, a region of genes, a region of amino acids, a region of DNAs, a region of proteins, all of them contained in the initial region represented by the cell. We are already faced with a necessary extension of the relation "contained in", used in the definition of a P system. Besides its usual meaning, when we refer, for instance, to the fact that DNA is included in the cell, we consider also the substring–string relation, as a variant of "contained in", accepting so that the region of nucleotide bases is contained in the region of DNA (meaning that any element of the former region is a substring of an element of the latter); similarly, the region of genes is contained, in this acception, in the region of DNAs; the region of amino acids is contained in the region of proteins, while the region of codons is contained in the region of RNAs and all are contained in the cell. While the DNA and RNA regions and their sub-regions, as they were shown above, are composed by strings over the input alphabet of the four types of nucleotide bases, proteins and various types of sub-proteins, such as cistrons, lead to regions and sub-regions whose elements are strings or substrings over the output alphabet of the 20 types of amino acids.

Another aspect deserving a reconsideration is the interior-exterior distinction, involved in the structure of a P system. In the light of the ideas exposed in the first section, it should be replaced by a four-steps organization: interior, exterior interior, interior exterior, and exterior, according to Hoffmeyer's approach. This means that some formal rules should be identified in order to distinguish a living system from its Umwelt, its Umwelt from its semiotic niche – sometimes called the *ecological niche* – and the ecological niche from its environment.

## 4 A Difficult Task: The Developmental Rules

The most difficult task is to identify the developmental rules associated with the considered regions. Just in this respect, the work done within the framework of HGP is essential [6]. The rules leading to the decomposition of DNA in nucleotide bases are of a chemical nature and recall the phonemic segmentation problem in descriptive linguistics. The rules identifying the genes are a mixture of chemistry, biology, and combinatorial and comparative operations; they lead

to approximations and to probabilistic statements, rather than to deterministic exact statements. In this respect, a measure of similarity between strings over the input alphabet or over the output alphabet is needed. This task is fulfilled by the introduction of the notion of an alignment of a pair of strings $\langle x, y\rangle$ as a new pair $\langle x', y'\rangle$, where $x'$ and $y'$ have the same length and $x'$ is obtained from $x$ and $y'$ is obtained from $y$ by inserting occurrences of the special space symbol –. For instance, if $x = acbcdb$ and $y = abbdcdc$, then a possible alignment of $x$ and $y$ is given by

$$x' = a\text{–}cbc\text{–}db,$$
$$y' = ab\text{–}bdcdc.$$

A symmetric scoring function $f$ is defined, that maps pairs of symbols from the alphabet $\{a, b, c, d, -\}$ to the real numbers. In respect of $f$, each individual column from the eight columns appearing in the representation of $x'$ and $y'$ has a score. The total score of the considered alignment is, by definition, the sum of scores for all columns; it expresses the similarity between $x$ and $y$ in respect to $f$. In order to make the total score, as an adequate measure of similarity between $x$ and $y$, it is necessary to make a right choice of the mapping $f$. For instance, it is natural to select for $f(\langle u, u\rangle)$ a value strictly higher than zero, for any object $u$ different from – in the alphabet, because matched symbols must increase the score of the alignment. However, one takes $f(\langle -, -\rangle) = 0$. It is also natural to oblige $f(\langle u, -\rangle)$ to be strictly negative, for any symbol $u$ different from –, in order to penalize misalignments. When $a, b$ are amino acids, $f(\langle a, b\rangle)$ indicates the frequency with which $a$ replaces $b$ in evolutionarily related strings. The global alignment problem asks for the optimal alignment of two strings $x$ and $y$ in respect to a given scoring function. As a measure of similarity, *optimal* means here the highest value possible, in contrast with other measures, which are looking for the size of dissimilarity, such as the Hamming distance.

For strings which are not globally similar, a kind of local alignment is investigated, which is weaker than the global one. The idea is to look for the alignment between consecutive substrings, chosen as desired, of $x$ and $y$. From two strings one can move to $n$ strings and define their multiple alignment, the score being the sum of the scores of the induced pairwise aligments. The problem to find a maximum-score multiple alignment of a set of strings is **NP**-hard (it is not the case for $n = 2$). See Karp [6: 547] for more details.

## 5 Exons, Introns, and Codons

Since gene finding and determining the functions of the proteins they encode were a basic task of HGP, discovering the rules in the P system accounting for them appears to be important. In this respect, we should perhaps distinguish between prokaryotes (whose cells do not have a distinct nucleus) and eukaryotes, whose cells include nuclei and organelles. In the former case, each gene consists of a single contiguous string of nucleotide bases. Things are more interesting

in higher eukaryotes, where a gene consists of two or more substrings called exons, that code for parts of a protein; exons are separated by introns, which are noncoding substrings. Some rules in the hypothetical associated P system should register the process of alternative splicing: the different possibilities of parsing a gene into exons and introns and the way the same gene can code for different proteins. In the transcription process from DNA to RNA, the string of exons and introns is transcribed into a pre-mRNA transcript, after which the introns are removed and the exons are spliced together to form the mRNA that leads to a ribosome, which in its turn is translated into protein.

Should exons and introns be included in the input alphabet of the P system and amino acids and protein functions into the output alphabet? Should these types of entities lead to different regions of the P system? Obviously, the input-output distinction corresponds here to the syntactic–semantic distinction, where *syntactic* is related to processes anterior to RNA–protein translation, while *semantic* is related to proteins and their functions. The input alphabet should also include all the signals indicating the exon–intron boundaries, as well as the beginning of the first exon and the end of the last exon of a gene. The entities involved in this operation are 64 types of codons (substrings of length equal to three), which correspond by the genetic code (dictionary) to twenty types of amino acids. Like morphemes in natural languages, which can be lexical or grammatical, some codons (whose number is 61) code for different types of amino acids, while three of them (TAA, TAG, TGA) are stop codons; they indicate the end of the translation process. The codon ATG is both lexical and grammatical, depending on its position; it may code an amino acid, but also the start of an exon. The complete picture of distribution of codons within exons and introns and of the distribution of nucleotide bases in respect to exon–intron boundaries is a mixture of deterministic and statistic aspects. The dynamic-programming Viterby algorithm gives the most likely evolution [6].

## 6 Phylogenetic Trees

Besides the problem of the structure of genomic sequences there is the problem of the evolution in time of a genetically related group of organisms. A P system depending on the parameter time should account for this process, where we are dealing with a phylogenetic tree whose leaves represent the existant species, while the internal nodes represent some postulated speciation events in which a species divides into populations that follow separate evolutionary paths and become distinct species. Karp [6: 540] makes clear the difference in aproaching the evolutionary tree before and after the era of genomics. Before this era, each species was described by means of some morphological characteristics, such as presence or absence of hair, fur, number and type of teeth, etc. Within the framework of genomics, the trees are mainly constructed by comparison of related DNA or protein sequences in the considered species, where for each species and each character a character state is given. Under this second aspect should phylogenetic trees be considered in the P systems perspective. Species with sim-

ilar character states should be close together in the tree. We reach in this way a problem of optimisation of a distance in the tree. Irrespective the way in which this optimisation problem is formulated, it proves to be **NP**-hard [6]. For instance, one can define the distance between two species as the sum of the lengths of the edges on the path between the two species in the tree. This fact gives rise to an opposite problem: Given a distance function $d$ defined on pairs of species, construct a corresponding tree and a set of edge distances, such that the resulting distance approximates $d$ as closely as possible. Phylogenetic trees should form a distinct region in the associated P system, while distances and their optimisation in these trees should code significant facts in the output alphabet.

We are only at the first steps of a problem that could involve a lot of technical difficulties.

## 7 Reads and Clones

A special attention deserve those fragments of DNA called *reads* and by means of which researchers in genomics try to approximate the structure of the whole genome, seen as a concatenation of reads. Their P system status is an open question. It is also interesting to pay attention to the fact that, according to a specific strategy, the sequencing of the entire genome is reduced to the sequencing of some fragments called *clones* (of length about 130,000 nucleotide bases); so, clones lead to another sub-region, whose elements are strings over the input alphabet. Let us point out also the holographic structure of the genome, as it is shown by the fact that each cell in our organism contains a copy of our whole genome. This huge apparent redundancy of the mechanism of our heredity may have nice implications for the associated P system.

## References

1. J. Aguado, T. Bălănescu, T. Cowling, M. Gheorghe, M. Holcombe, F. Ipate, P systems with replicated rewriting and stream Eilenberg machines, *Fundamenta Informaticae*, 49, 1-3 (2002), 17–33.
2. G.J. Chaitin, Meta-mathematics and the foundations of mathematics, *Bulletin of the EATCS*, 77 (June 2002), 167–179.
3. C. Emmeche, K. Kuhl, F. Stjernfelt, Reading Hoffmeyer, rethinking biology, *Tartu Semiotic Library* 3, Tartu University Press, 2002.
4. J. Hoffmeyer, Surfaces inside surfaces, *Cybernetics and Human Knowing*, 5, 1 (1998), 33–42.
5. J. Hoffmeyer, The biology of signification, *Perspectives in Biology and Medicine*, 43, 2 (2000), 252–268.
6. R.M. Karp, Mathematical challenges from genomics and molecular biology, *Notices of the American Mathematical Society*, 49, 5 (May 2002), 544–553.
7. Gh. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
8. J. Uexküll, The theory of meaning, *Semiotica*, 42, 1 (1982) [1940], 25–82.
9. S. Wolfram, *A New Kind of Science*, Wolfram Media Inc., 2001.

# Probabilistic P Systems

Adam Obtułowicz

Institute of Mathematics
Polish Academy of Sciences
Śniadeckich 8, P.O. Box 137
00-950 Warsaw, Poland
adamo@impan.gov.pl

**Abstract.** There are introduced and discussed stochastic and randomized P systems. Stochastic P systems are aimed to describe distortions which may appear in evolution processes of membrane systems considered in the area of membrane computing. Randomized P systems serve for implementation of randomized algorithms. There is presented a family of randomized P systems which serve for implementation of Miller–Rabin randomized algorithm for primality of integers. P systems of this family verify primality of integers in a polynomial time, with a low error probability, and with a subexponential number of processors modelled as membranes with evolving contents.

## 1 Introduction

Our purpose is to answer a question about a possible probabilistic approach to P systems, see question 26 formulated by G. Păun in P systems Web Page at http://psystems.disco.unimib.it/problems.html, and the question concerning the implementation of randomized algorithms in P systems, see [8].

We introduce the following two new classes of P systems:

- stochastic P systems, aimed to describe in a statistical manner certain distortions (delays) which may appear at random in possible physical (biochemical) realizations of P systems,
- randomized P systems, used to implement randomized algorithms.

Stochastic P systems are defined in the manner of stochastic Petri nets described in [6]. We notice here that stochastic Petri nets besides being used in modeling various systems in computer science, see [2], are also applied in molecular biology, see, for instance, [3]. Briefly, stochastic P systems are P systems equipped with average delays of applications of evolution rules in a similar way as stochastic Petri nets are Petri nets equipped with average delays of firing of transitions.

Randomized P systems are aimed to decrease the exponential expansion of the number of membranes which appear in processes generated by P systems used to solve NP-problems in a polynomial time and described in [7], [8], [10]. A decreasing of the exponential expansion of the number of membranes to some

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 377–387, 2003.

subexponential one is achieved with a loss of certainty of a final result which is reached with some error probability in a similar way as in the case of randomized algorithms, where a subexponential time of computations is achieved with a loss of certainty of a final result, see [5], [9]. Randomized P systems are similar to those discussed in [7], [8], [10] except that the initial configurations (beginning processes of evolution) of randomized P systems contain certain multisets of objects chosen at random. These randomly chosen configurations of objects steer the processes of evolution in such a way that a subexponential number of membranes appear in the process.

Randomized P systems serve for implementation of randomized algorithms. We present in the paper a family of randomized P systems which serve for implementation of Miller–Rabin randomized algorithms for primality of integers. P systems of this family verify primality of integers in a polynomial time, with a low error probability, and with a subexponential number of membranes contained in membrane systems appearing in processes generated by these P systems.

One finds that randomness requiring some portion of probability Tteory to describe it appears in stochastic and randomized P systems. Therefore we propose to call probabilistic P systems those P systems which are stochastic or randomized.

For all unexplained terms and notation concerning P systems and multisets we refer the reader to [8].

## 2 Stochastic P Systems

Before introducing stochastic P systems we show that to every Petri net $\mathcal{N}$ one can assign a one membrane P system $\Pi(\mathcal{N})$, where the applications of evolution rules of $\Pi(\mathcal{N})$ coincide with the firing of transitions of $\mathcal{N}$ such that the processes generated by $\Pi(\mathcal{N})$ can be identified with the processes generated by $\mathcal{N}$ with respect to the length of processes and processed objects. Then we define stochastic P systems in such a way that the assignment of P systems $\Pi(\mathcal{N})$ to Petri nets $\mathcal{N}$ can be simply extended to an assignment from the class of stochastic Petri nets as considered in [6] into the class of stochastic P systems, where firing of transitions still coincide with applications of evolution rules.

We recall that a *Petri net* is an ordered quintuple $\mathcal{N} = (P_{\mathcal{N}}, T_{\mathcal{N}}, \mathrm{Pre}_{\mathcal{N}}, \mathrm{Post}_{\mathcal{N}}, \mathcal{M}_0)$, where $P_{\mathcal{N}}$ and $T_{\mathcal{N}}$ are the *sets of places* and *transitions* of $\mathcal{N}$, respectively, the sets $\mathrm{Pre}_{\mathcal{N}} \subset P_{\mathcal{N}} \times T_{\mathcal{N}}$, $\mathrm{Post}_{\mathcal{N}} \subset T_{\mathcal{N}} \times P_{\mathcal{N}}$ are the *sets of input arcs* and *output arcs* of $\mathcal{N}$, respectively, and $\mathcal{M}_0$ is a function (multiset), called *initial marking of* $\mathcal{N}$, which is defined on $P_{\mathcal{N}}$ and valued in the set $\mathbb{N}$ of natural numbers. For two functions (multisets) $\mathcal{M} : P_{\mathcal{N}} \to \mathbb{N}$, $\mathcal{M}' : P_{\mathcal{N}} \to \mathcal{N}$, called *markings*, and a transition $t \in T_{\mathcal{N}}$ we say that $\mathcal{M}'$ is a *result of firing* $t$ *for* $\mathcal{M}$, briefly writing $\mathcal{M}[t\rangle\mathcal{M}'$, if $\varphi_{\mathrm{Pre}(t)} \leq \mathcal{M}$ and $\mathcal{M}' = (\mathcal{M} \dot{-} \varphi_{\mathrm{Pre}(t)}) + \varphi_{\mathrm{Post}(t)}$, where $\varphi_{\mathrm{Pre}(t)}$ and $\varphi_{\mathrm{Post}(t)}$ are characteristics functions of sets $\{p \in P_{\mathcal{N}} \mid (p,t) \in \mathrm{Pre}_{\mathcal{N}}\}$ and $\{p \in P_{\mathcal{N}} \mid (t,p) \in \mathrm{Post}_{\mathcal{N}}\}$, respectively.

To a Petri net $\mathcal{N} = (P_{\mathcal{N}}, T_{\mathcal{N}}, \mathrm{Pre}_{\mathcal{N}}, \mathrm{Post}_{\mathcal{N}}, \mathcal{M}_0)$ with $P_{\mathcal{N}} \cap T_{\mathcal{N}} = \varnothing$ we assign a one membrane P system $\Pi(\mathcal{N}) = \big(\mathcal{S}(\mathcal{N}), \mathcal{R}(\mathcal{N})\big)$ such that $\mathcal{S}(\mathcal{N})$ is a membrane system containing one membrane $[\ ]$ and:

- the labelling function $l : \{[\ ]\} \to \{1\}$ is given by $l([\ ]) = 1$ and the electric charge function $e : \{[\ ]\} \to \{-, 0, +\}$ is given by $e([\ ]) = 0$,
- the set $O(\mathcal{N})$ of objects is defined by

$$O(\mathcal{N}) = P_{\mathcal{N}} \cup T_{\mathcal{N}} \cup \{@\}, \text{ for } @ \notin P_{\mathcal{N}} \cup T_{\mathcal{N}},$$

- the labelling function $M : \{[\ ]\} \to \mathbb{N}^{O(\mathcal{N})}$ is given by

$$M\big([\ ]\big)(x) = \begin{cases} \mathcal{M}_0(x) & \text{if } x \in P_{\mathcal{N}}, \\ 1 & \text{otherwise.} \end{cases}$$

The set $\mathcal{R}(\mathcal{N})$ of evolution rules is the set of rules of the following form

$$\big[_1\ @tu_t \to @tw_t\ \big]_1^0,$$

for $t \in T_{\mathcal{N}}$ and for some strings $u_t$ and $w_t$ of objects of $\mathcal{S}(\mathcal{N})$ which are chosen in a unique way for $t$ provided $u_t$ and $w_t$ present characteristic functions $\varphi_{\mathrm{Pre}(t)}$ and $\varphi_{\mathrm{Post}(t)}$, respectively. Thus @ should be present in the system.

A rule of the above form is called a rule *induced* by a transition $t$.

**Theorem 1.** *For a Petri net $\mathcal{N}$ with $P_{\mathcal{N}} \cap T_{\mathcal{N}} = \varnothing$, if the P system $\Pi(\mathcal{N})$ assigned to $\mathcal{N}$ generates a process*

$$S_0 \overset{\mathcal{P}_1}{\Longrightarrow} S_1 \overset{\mathcal{P}_2}{\Longrightarrow} S_2 \ldots S_{n-1} \overset{\mathcal{P}_n}{\Longrightarrow} S_n,$$

*then the membrane systems $S_0, S_2, \ldots, S_n$ contain exactly one membrane $[\ ]$, the sets $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$ of places of application of evolution rules are one element sets, and the labelling functions $M_0, M_1, \ldots, M_{n-1}, M_n$ valued in $\mathbb{N}^{O(\mathcal{N})}$ of membrane systems $S_0, S_2, \ldots, S_{n-1}, S_n$, respectively, are such that for every $i$ with $0 \leq i \leq n-1$ we have that*

$$M_i\big([\ ]\big) \restriction P_{\mathcal{N}}[t_{i+1}\rangle M_{i+1}\big([\ ]\big) \restriction P_{\mathcal{N}},$$

*where $t_{i+1}$ is a transition of $\mathcal{N}$ such that for a unique place of application $p \in \mathcal{P}_{i+1}$ the rule $r_p$ is a rule induced by $t_{i+1}$ and $M_i\big([\ ]\big) \restriction P_{\mathcal{N}}$, $M_{i+1}\big([\ ]\big) \restriction P_{\mathcal{N}}$ are the restrictions of $M_i\big([\ ]\big)$ and $M_{i+1}\big([\ ]\big)$ to the set $P_{\mathcal{N}}$, respectively. Analogous conditions hold also for infinite processes generated by $\Pi(\mathcal{N})$.*

*Proof.* Since by definition of application of evolution rules two different rules cannot be applied simultaneously to the same object, the definition of $\mathcal{R}(\mathcal{N})$ provides that $\mathcal{P}_i$ $(1 \leq i \leq n)$ are one element sets (see the occurrence of object @ in the rules), where a unique place of application $p \in \mathcal{P}_i$ is such that the rule $r_p$ is induced by a transition fired in the $i$-th step of a process generated by $\mathcal{N}$. □

Thus, one can treat the notion of a P system as a generalization of the notion of a Petri net, where applications of evolution rules coincide with firing of transitions.

In [6] a *stochastic Petri net* is defined as a Petri net $\mathcal{N}$ equipped with a mapping $d_{\mathcal{N}} : T_{\mathcal{N}} \to \mathbb{R}$ valued in the set $\mathbb{R}$ of real numbers, where for every transition $t \in T_{\mathcal{N}}$ the value $d_{\mathcal{N}}(t)$ is a *firing rate associated with* $t$, i.e., $\big(d_{\mathcal{N}}(t)\big)^{-1}$ is the average firing delay of transition $t$ (resulting from experiments).

Since the notion of a P system can be treated as a generalization of the notion of a Petri net, we propose the following definition of a stochastic P system.

A *stochastic* P *system* is a P system $\Pi = (\mathcal{S}, \mathcal{R})$ equipped with a mapping $d_\Pi : \mathcal{R} \to \mathbb{R}$, where for every rule $r \in \mathcal{R}$ the value $\big(d_\Pi(r)\big)^{-1}$ is interpreted as the average delay of an application of $r$.

Thus the assignment of P systems $\Pi(\mathcal{N})$ to Petri nets $\mathcal{N}$ can be extended to an assignment from the class of stochastic Petri nets to the class of stochastic P systems.

## 3 Randomized P Systems

Randomized P systems are aimed to implement randomized algorithms. Before introducing a proposal of a randomized P system, we explain the concept of a randomized algorithm by discussing the well-known Miller–Rabin randomized algorithm for primality of integers described among others in [1] and [4].

Miller–Rabin randomized algorithm for primality of integers is aimed to verify whether an odd natural number $n$ is prime. This algorithm contains the *strong Fermat test for* $n$ *to the base* $a$ performed for a randomly chosen integer $a$ in the range $1 < a < n - 1$, see [4]. The above strong Fermat test is a deterministic algorithm given in the form of program in [1], see Miller–Rabin($n$) algorithm on page 282 in that book, where deleted instruction (1), of a random choice of the parameter $a$, is deleted. The following theorem describes properties of the test.

**Theorem 2.** *If $n > 2$ is a prime number, then it passes the strong Fermat test to the base $a$ for all natural numbers $a$ in the range $1 < a < n - 1$. If $n$ is an odd composite number, then it does not pass the strong Fermat test to the base $a$ for at least $3/4$ of all natural numbers $a$ in the range $1 < a < n - 1$. Strong Fermat test for $n$ to the base $a$ is performed in polynomial time with respect to the length of binary presentation of $n$ for every natural number $a$ in the range $1 < a < n - 1$.*

*Proof.* The proof is analogous to the proof of Theorem 9.4.5 in [1]. □

**Corollary 1.** *If the strong Fermat test for $n$ to the base $a$ is performed for $k$ different randomly chosen values of the base $a$ in the range $1 < a < n - 1$ with $1 \leq k < \frac{n-1}{4}$ and $n$ passes the test for all these $k$ randomly chosen values of $a$, then the error probability of the test, i.e., the probability that $n$ is not prime when $n$ passes the test, is bounded above by $(1/4)^k$.*

*Proof.* The assertion is an immediate consequence of Theorem 2. □

Therefore, the final result of the application of Miller–Rabin randomized algorithm for primality of integers to an odd natural number $n$ is uncertain and reached with some error probability as described in Corollary 1 when $n$ passes the strong Fermat test for $k$ different randomly chosen values of the base $a$ in the range $1 < a < n-1$ with $1 \leq k < \frac{n-1}{4}$. We should notice that if $n$ does not pass the test for some value of $a$, then $n$ is definitely a composite number, see Theorem 2.

Thus one can describe informally a *randomized algorithm* as an algorithm which contains a possibly deterministic algorithm (test) performed for some randomly chosen input data in a possibly polynomial time, where some estimation of error probability of the final result of the test is known; for a more formal approach, see [9]. Randomized algorithms are used when for their tasks there are not known efficient deterministic algorithms.

Starting from the Miller–Rabin randomized algorithm for primality of integers, we explain an idea of randomized P systems used to implement a randomized algorithm.

One can use P systems $\Pi^{(n)}$ introduced in [8] for checking primality of integers in a polynomial time but these P systems generate processes where there appear membrane systems containing an exponential number of membranes with respect to the length of binary presentation of a natural number $n$ to be checked for primality. In order to decrease this exponential number of membranes to some subexponential one we propose another approach related to the methods of randomized algorithms, where we use randomized P systems. The subexponentiality is understood in the following way. For the length $k$ of input data, the time or space complexity measures with respect to $k$ of a given algorithm are called *subexponential* if they are bounded by $e^{f(k)}$ for some function $f(k)$ with $\lim\limits_{k\to\infty} \frac{f(k)}{k} = 0$, see [4].

We use the following auxiliary notions.

We say that an ordered quadruple $(k, s, s', s'')$ consisting of a natural number $k$ and three binary strings $s = \sigma_1 \dots \sigma_m$, $s' = \sigma'_1 \dots \sigma'_{m'}$, $s'' = \sigma''_1 \dots \sigma''_{m''}$ of length $m$, $m'$, $m''$, respectively, is a *trap* if the following conditions hold:

1. $m = m'$ and $2 < m - m'' = k < m$,
2. both the first and the last element of $s$ is $\mathbf{1}$ and $\mathbf{1}$ occurs $k$ times in $s'$,
3. for a binary string $s''' = \sigma'''_1 \dots \sigma'''_m$ satisfying conditions $(\alpha)$ and $(\beta)$ if $n$ and $n'''$ are natural numbers presented binary by $s$ and $s'''$, respectively (up to meaningless occurrences of zeroes), then $n''' < n - 1$, where the conditions $(\alpha)$ and $(\beta)$ are given in the following way:
   $(\alpha)$ for all $i$ with $1 \leq i \leq m$ if $\sigma'_i = \mathbf{0}$, then $\sigma'''_i = \sigma''_j$ for $j = i - k_i$, where $k_i$ is the number of elements of $\{p \mid \sigma'_p = \mathbf{1} \text{ and } p < i\}$,
   $(\beta)$ for all $i$ with $1 \leq i \leq m$ if $\sigma'_i = \mathbf{1}$, then $\sigma'''_i = \mathbf{1}$.

By the *scope* of a trap $(k, s, s', s'')$ with $s' = \sigma'_1 \dots \sigma'_m$ and $s'' = \sigma''_1 \dots \sigma''_{m''}$ we mean the set of binary strings $s''' = \sigma'''_1 \dots \sigma'''_m$ of length $m$ such that the above condition $(\alpha)$ holds and $1 < n'''$ for $n'''$ binary presented by $s'''$.

**Lemma 1.** *The scope of a trap* $(k, s, s', s'')$ *has at least* $2^k - 2$ *elements and at most* $2^k$ *elements. Every binary string* $s'''$ *belonging to the scope of a trap* $(k, s, s', s'')$ *presents (up to meaningless occurrences of zeroes) a natural number* $n'''$ *such that* $1 < n''' < n - 1$*, where* $n$ *is the number presented binary by the string* $s$*.*

*Proof.* This lemma is an immediate consequence of the definitions of a trap and its scope. □

We introduce now P systems which can be treated as P systems serving for implementation of Miller–Rabin randomized algorithm for primality of integers. We use the strong Fermat test for $n$ to the base $a$ written in the form of a liberal loop program $\mathcal{P}'_{\mathrm{sFt}}$ given in the Appendix of the present paper.

For a trap $(k, s, s', s'')$ with $s = \sigma_1 \dots \sigma_m$, $s' = \sigma'_1 \dots \sigma'_m$, $s'' = \sigma''_1 \dots \sigma''_{m''}$ we define a P system $\mathit{\Pi}[k, s, s', s''] = (\mathcal{S}[k, s, s', s''], \mathcal{R}[k, s, s', s''])$ as follows. The initial membrane system $\mathcal{S}[k, s, s', s'']$ is defined for the $l$-program $\mathcal{P}'_{\mathrm{sFt}}$ in the following way:

— the membrane system $\mathcal{S}[k, s, s', s'']$ is such that its underlying set $\mathbb{B}$ of balls contains exactly three balls $b_2 \subsetneq b_1 \subsetneq b_0$, the set $L$ of labels of $\mathcal{S}[k, s, s', s'']$ is the set $\{0, 1, 2\}$, and $l(b_i) = i$, $e(b_i) = 0$ for all $i \in \{0, 1, 2\}$,

— the set $O$ of objects of $\mathcal{S}[k, s, s', s'']$ is the set of ordered triples $(x, y, z)$, written $(x)^y_z$, such that

- $x \in U = \{\mathbf{0}, \mathbf{1}, \triangleright\} \cup \{0, 1, \dots, |\mathcal{P}'_{\mathrm{sFt}}| + 1\}$, where $|\mathcal{P}'_{\mathrm{sFt}}|$ denotes the length of program $\mathcal{P}'_{\mathrm{sFt}}$ treated as a sequence of instructions,
- $y \in W = \mathrm{RGR}(\mathcal{P}'_{\mathrm{sFt}}) \cup \{!, ?, -@, @, +@, \bot\}$, where $\mathrm{RGR}(\mathcal{P}'_{\mathrm{sFt}})$ denotes the set of register names which occur in $\mathcal{P}'_{\mathrm{sFt}}$ and the digits $\mathbf{0}, \mathbf{1}$ together with auxiliary symbols $!, ?, -@, @, +@, \triangleright, \bot$ are different from natural numbers and names of registers in $\mathrm{RGR}(\mathcal{P}'_{\mathrm{sFt}})$,
- $z \in \{0, 1, \dots, m + 1\} \cup \{\bot, @\}$ for $x \in U$ and $y \in W - \{!, ?\}$,
  $z \in \{1, \dots, m - 1\}$ for $x \in \{\mathbf{0}, \mathbf{1}\}$ and $y \in \{!, ?\}$,
  $z = 0$ for $x = \triangleright$ and $y \in \{!, ?\}$,

— the labelling function $M : \mathbb{B} \to \mathbb{N}^O$ valued in the set $\mathbb{N}^O$ of multisets is given by

$$M(b_0) = M(b_1) = \mathbb{O},$$

where $\mathbb{O}(a) = 0$ for every object $a \in O$,

$$\begin{aligned} M(b_2) = {} & \left(\sum_{i=1}^{m} \left\langle (\sigma_i)^N_i \right\rangle\right) + \left\langle (\triangleright)^N_{m+1} \right\rangle + \\ & \left(\sum_{i=1}^{m-1} \left\langle (\sigma^!_i)^!_i \right\rangle\right) + \left\langle (\triangleright)^!_0 \right\rangle + \left\langle (\sigma'_m)^{-@}_m \right\rangle + \\ & \left(\sum_{j=1}^{m-k-1} \left\langle (\sigma''_j)^?_j \right\rangle\right) + \left\langle (\triangleright)^?_0 \right\rangle + \left\langle (\sigma''_{m-k})^{+@}_{m-k} \right\rangle + \\ & \sum_{z \in I^-} \left\langle (\triangleright)^Z_1 \right\rangle, \end{aligned}$$

where $A$ and $N$ are input registers of $\mathcal{P}'_{\mathrm{sFt}}$ and $I^- = \mathrm{RGR}(\mathcal{P}'_{\mathrm{sFt}}) - \{A, N\}$.

The set $\mathcal{R}[k, s, s', s'']$ of evolution rules contains the rules given by the following schemes:

$(S'_1)$ $[_2(\mathbf{1})_i^{-@} \to (\rhd)_i^{@}\,]_2^0$ for $1 \le i \le m$,
$(S'_2)$ $[_2(\rhd)_i^{@}\,]_2^0 \to [_2(\rhd)_i^{-@}\,]_2^- [_2(\rhd)_i^{+@}\,]_2^+$ for $1 \le i \le m$,
$(S'_3)$ $[_2(\delta)_{i-1}^{!}(\rhd)_i^{-@} \to (\delta)_{i-1}^{-@}(\mathbf{0})_i^A\,]_2^-$ for $1 \le i \le m$ and $\delta \in \{\mathbf{0}, \mathbf{1}, \rhd\}$,
$(S'_4)$ $[_2(\delta)_{i-1}^{!}(\rhd)_i^{+@} \to (\delta)_{i-1}^{+@}(\mathbf{1})_i^A\,]_2^+$ for $1 \le i \le m$ and $\delta \in \{\mathbf{0}, \mathbf{1}, \rhd\}$,
$(S'_5)$ $\big[_1\,[_2\,]_2^-\,[_2\,]_2^+\,\big]_1^0 \to \big[_1\,[_2\,]_2^0\,\big]_1^0\,\big[_1\,[_2\,]_2^0\,\big]_1^0$,
$(S'_6)$ $[_2(\delta)_{i-1}^{!}(\delta')_{j-1}(\mathbf{0})_i^{-@}(\delta'')_j^{+@} \to (\delta)_{i-1}^{-@}(\delta')_{j-1}^{+@}(\delta'')_i^A\,]_2^0$ for $1 \le i \le m$, $1 \le j \le m-k$, $\delta, \delta' \in \{\mathbf{0}, \mathbf{1}, \rhd\}$, $\delta'' \in \{\mathbf{0}, \mathbf{1}\}$,
$(S'_7)$ $[_2(\rhd)_0^{-@}(\rhd)_0^{+@} \to (\rhd)_{m+1}^A(1)_{@}^{\perp}\,]_2^0$ which initializes the simulation of execution of program $\mathcal{P}'_{\mathrm{sFt}}$,
$(P')$ the set of evolution rules determined by the $l$-program $\mathcal{P}'_{\mathrm{sFt}}$ which is defined in an analogous way as the set of evolution rules determined by the $l$-program $\mathcal{P}_{\mathrm{fct}}$ in [8],
$(F'_1)$ $[_2(|\mathcal{P}'_{\mathrm{sFt}}| + 1)_{@}^{\perp}(\rhd)_2^C \to (1)_{\perp}^{\perp}\,]_2^0$, where $C$ is the output register of $\mathcal{P}'_{\mathrm{sFt}}$,
$(F'_2)$ $[_i(1)_{\perp}^{\perp}\,]_i^0 \to [_i\,]_i^+(1)_{\perp}^{\perp}$ for $i \in \{0, 1, 2\}$,

where the rules $(F'_1)$ and in $(F'_2)$ enable to read the final result.

There is no any other rule in $\mathcal{R}[k, s, s', s'']$ than those described by the above schemes.

**Theorem 3.** *For every trap $(k, s, s', s'')$, the P system $\Pi[k, s, s', s'']$ generates a single evolution process of a polynomial length with respect to the length of $s$ such that the outermost membrane of the last membrane system of the process has electric charge 0 iff the number $n$ presented binary by $s$ passes the strong Fermat test to the base $a$ for every value of $a$ whose binary presentation (up to the meaningless occurrences of zeroes) is in the scope of the trap $(k, s, s', s'')$. If this outermost membrane has electric charge $+$, then the number $n$ does not pass the strong Fermat test to the base $a$ for some value of $a$ with $1 < a < n - 1$. Moreover, the membrane systems appearing in the process contain at most $2^k$ membranes.*

*Proof.* The proof is similar to the proof of the main theorem of Section 3 in [8]. Here in $2^k$ membranes obtained by the application of rules $(S'_1)$–$(S'_6)$ one simultaneously simulates the realizations of the $l$-program $\mathcal{P}'_{\mathrm{sFt}}$ for $2^k$ different input data, respectively during the evolution process. These $2^k$ different input data are strings belonging to the scope of the trap $(k, s, s', s'')$ except at most two strings (presenting 0 and 1, respectively) which are rejected by the first instruction of $\mathcal{P}'_{\mathrm{sFt}}$. Numerical data are represented by strings $(\delta_1)_1^X, \ldots, (\delta_n)_n^X$ of objects contained in membranes for $\delta_1, \ldots, \delta_n \in \{\mathbf{0}, \mathbf{1}\}$ and a register $X \in \mathrm{RGR}(\mathcal{P}'_{\mathrm{sFt}})$. □

We say that a P system $\Pi[k, s, s', s'']$ *accepts* the number presented binary by $s$ if the outermost membrane of the last membrane system in the process generated by $\Pi[k, s, s', s'']$ has electric charge 0.

A P system $\Pi[k, s, s', s'']$ is called a *randomized* P system if $s'$ and $s''$ are randomly chosen binary strings.

Until there is not known a statistical dependence between:

- randomness of elements of the scope of a trap $(k, s, s', s'')$ for randomly chosen binary strings $s'$ and $s''$,
- randomness of an error of a test for primality of the number $n$ presented by $s$, where we mean that $n$ passes the test iff P system $\Pi[k, s, s', s'']$ accepts $n$,

one may accept the following hypothesis by virtue of Theorems 2, 3, and of Lemma 1.

*Hypothesis.* If a randomized P system $\Pi[k, s, s', s'']$ accepts the number $n$ presented binary by $s$, then $n$ is prime with an error probability bounded above by $16 \cdot (4)^{-2^k}$

Thus by Theorem 3 and by assuming the above Hypothesis, a family of randomized P systems $\Pi[f(|s|), s, s', s'']$ with $\lim\limits_{n\to\infty} \frac{f(n)}{n} = 0$ provides verification of primality of integers with a low error probability (bounded above by $16 \cdot (4)^{-2^{f(|s|)}}$) in a polynomial time and with a subexponential number (bounded above by $2^{f(|s|)}$) of membranes contained in membrane systems appearing in the process generated by the P systems $\Pi[f(|s|), s, s', s'']$, where $|s|$ denotes the length of $s$.

We treat a family of randomized P systems $\Pi[f(|s|), s, s', s'']$ with $\lim\limits_{n\to\infty} \frac{f(n)}{n}$ equal to 0 as an implementation of Miller–Rabin randomized algorithm for primality of integers in randomized P systems. Thus we propose the following general definition.

A family of randomized P systems *serves for implementation* of a given *randomized algorithm* containing a test for some randomly chosen input data if some configuration of objects in the initial membrane systems of P systems are such that:

- they represent some randomly chosen data, see for instance the formula defining $M(b_2)$ for $\mathcal{S}[k, s, s', s'']$,
- they steer the evolution process to reach some possibly subexponential number $x$ of membranes in which there are simultaneously simulated performances of the test (contained in a given randomized algorithm) for $x$ different randomly chosen input data for the test, respectively.

We do not exclude other approaches to modeling randomized P systems which can serve for implementation of randomized algorithms.

*Open Problem (tossing* $\mathbf{0}$, $\mathbf{1}$ *by P systems).* One sees that in the case of randomized P systems $\Pi[k, s, s', s'']$ the applications of evolution rules $(S'_1)$–$(S'_6)$ do not give $2^k$ different statistically independent randomly chosen integers $a$ (with $1 < a < n-1$) represented by strings $(\delta_1)^A_1, \ldots, (\delta_m)^A_m$ (with $\delta_1 \ldots, \delta_m \in \{\mathbf{0}, \mathbf{1}\}$) of objects contained in $2^k$ membranes, respectively, of evolving membrane system in the process generated by $\Pi[k, s, s', s'']$. In this case randomness of these

$2^k$ different integers $a$ with $1 < a < n-1$ is determined by random choices of strings $s'$, $s''$. Thus one can ask whether there exists a system of evolution rules providing unbiased tossing of **0**, **1** or there exists another configuration of objects in the initial membrane system than that given by the formula defining $M(b_2)$ such that one could get $2^k$ different statistically independent randomly chosen integers $a$ with $1 < a < n-1$ represented by strings of objects contained in $2^k$ membranes, respectively, of evolving membrane system.

## Appendix

A program $\mathcal{P}_{\mathrm{sFt}}$ for the strong Fermat test for $n$ to the base $a$ is written as a liberal loop program meant as in [8] and it is outlined in Fig. 1.

For unexplained terms and notation concerning liberal loop programs we refer the reader to [8]. We describe now the meaning of some expressions which appear in Fig. 1.

The program $\mathcal{P}_{\mathrm{sFt}}$ has two input registers $A$ and $N$, where before its execution $A$ is aimed to contain a binary presented value of the base $a$ with $1 < a < n-1$ and $N$ is aimed to contain a binary presented odd natural number $n$ being the subject of the strong Fermat test for $n$.

For the expression $1/$ stands an $l$-program with input register $N$ and two output registers $S, D$. This program computes a string $s$ of length $|s|$ and a binary presented odd natural number $d$ such that $n-1 = 2^{|s|} \cdot d$ for a binary presented odd natural number $n$ contained in the input register $N$. The string $s$ and the natural number $d$ are contained in the registers $S$ and $D$, respectively, after execution of the program $1/$.

For the instruction $2/$ stands an $l$-program with input registers $A, D, N$, and an output register $U$. This program computes the binary presented value of $a^d \pmod n$ for binary presented values $a, d, n$ contained in the input registers $A, D, N$, respectively. The value of $a^d \pmod n$ is contained in the output register $U$.

For the instruction $5/$ stands an $l$-program described in a similar way as in the case of instruction $2/$.

For the instruction $6/$ stands an $l$-program with input registers $K, N, S, U$, and an output register $T$. This program verifies whether both $(*)$ $|k| = |s|$ and $(**)$ $u \not\equiv 1 \pmod n$ hold for the string $k, s$, and binary presented natural numbers $u, n$ contained in the registers $K, S, U, N$, respectively, where $|k|, |s|$ denote the length of $k$ and $s$, respectively. When $(*)$ and $(**)$ hold, the output register $T$ contains one element string "**0**" after execution of the program $6/$, otherwise $T$ is cleaned, i.e., $T$ contains empty string $\Lambda$ after execution of $6/$.

For the instructions $3/$ and $7/$ stand $l$-programs described in a similar way as in the case of instruction $6/$.

The instruction $4/$ means to write the digit **1** on the right end of the current content of the register $K$.

The remaining instructions of $\mathcal{P}_{\mathrm{sFt}}$ are described as in [8].

```
1/  P[S and D are s.t. N − 1 = 2^|S| · D & D is odd];
2/  U := P[A power D (mod N)];
    K := Λ;
    LOOP|S|;
3/    T := P[U ≢ 1 (mod N)];
      LOOP|T|;
4/      K := K & 1;
        W := U;
5/      U := P[U power 2 (mod N)];
      END;
    END;
6/  T := P[|K| = |S| & U ≢ 1 (mod N)];
    LOOP|T|;
      C := 0;
    END;
    T := Λ ⊖ T;
    LOOP|T|;
      R := Λ ⊖ K;
      LOOP|R|;
        C := Λ;
      END;
      R := Λ ⊖ R;
      LOOP|R|;
7/      Q := P[W ≢ −1 (mod N)];
        LOOP|Q|;
          C := 0;
        END;
        Q := Λ ⊖ Q;
        LOOP|Q|;
          C := Λ;
        END;
      END;
    END;
```

**Fig. 1.**

Since the execution of instruction LOOP$|S|$ means the execution $|s|$ times of the program between the occurrence of LOOP$|S|$ and the occurrence of instruction END matching with the occurrence of LOOP$|S|$ for $|s|$ equal to the length of the current content of the register $S$, after execution of LOOP$|S|$ the registers $U$

and $W$ contain binary presented values of the numbers $a_k$ and $a_{k-1}$, respectively, which result from the performance of instruction (3) of Miller–Rabin algorithm in [1] p. 282.

The execution of the remaining part of $\mathcal{P}_{\mathrm{sFt}}$ following the occurrence of END matching with the occurrence of LOOP$|S|$ yields an analogous result to the result of performance of instructions (4)–(7) of Miller–Rabin algorithm in [1] p. 282. The output register $C$ of $\mathcal{P}_{\mathrm{sFt}}$ contains one element string "**0**" when $n$ does not pass the strong Fermat test to the base $a$, otherwise the register $C$ is cleaned.

For some practical reasons we use program $\mathcal{P}'_{\mathrm{sFt}}$ given by

$$
\begin{aligned}
&C := [A > 1];\\
&\mathrm{LOOP}|C|;\\
&\quad \mathcal{P}_{\mathrm{sFt}}\\
&\mathrm{END};
\end{aligned}
$$

where instruction $C := [A > 1]$ is executed in the following way. Register $C$ is cleaned when register $A$ contains a string of zeroes or a string with the last element **1** preceded by a string, maybe empty, of zeroes, otherwise one puts one element string "**0**" in $C$.

## References


1. E. Bach and J. Schallit, *Algorithmic Number Theory*, Cambridge, Massachusetts 1996.
2. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad, *Stochastic well-formed coloured nets for symmetric modeling applications*, IEEE Transactions on Computers Vol. 42, No. 11 (1993), pp. 1343–1360.
3. P.J. Goss and J. Peccoud, *Quantitative modelling of stochastic systems in molecular biology by using stochastic Petri nets*, Proc. Nat. Acad. Sci. USA Vol. 95, June 1998, pp. 6750–6755.
4. N. Koblitz, *Algebraic Aspects of Cryptography*, Berlin 1998.
5. A.K. Lenstra, H.W. Jr. Lenstra, *The Development of the Number Field Sieve*, Lecture Notes in Mathematics Vol. 1554, Berlin 1993.
6. M.A. Marsan, *Stochastic Petri Nets: An Elementary Introduction*, in Advances in Petri Nets 1989, ed. G. Rozenberg, Lecture Notes in Computer Science Vol. 424, Berlin 1990, pp. 1–29.
7. A. Obtułowicz, *Deterministic P systems for Solving SAT-problem*, Romanian Journal of Information Science and Technology Vol. 4, No. 1–2, pp. 195–201.
8. A. Obtułowicz, *On P systems with Active Membranes Solving the Integer Factorization Problem in a Polynomial Time*, in Multiset Processing, ed. C.S. Calude et al., Lecture Notes in Computer Science Vol. 2235, Berlin 2001, pp. 267–285.
9. Ch.H. Papadimitriou, *Computational Complexity*, Reading Massachusetts 1994.
10. G. Păun, *P systems with Active Membranes: Attacking NP Complete Problems*, Journal of Automata, Languages and Combinatorics 6 (2000), pp. 75–90.

# Decision P Systems and the P≠NP Conjecture

Mario J. Pérez Jiménez, Álvaro Romero Jiménez,
and Fernando Sancho Caparrini

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla, España
{Mario.Perez,Alvaro.Romero,Fernando.Sancho}@cs.us.es

**Abstract.** We introduce *decision* P systems, which are a class of P systems with symbol-objects and external output. The main result of the paper is the following: if there exists an NP–complete problem that cannot be solved in polynomial time, with respect to the input length, by a deterministic decision P system constructed in polynomial time, then $P \neq NP$. From Zandron-Ferreti-Mauri's theorem it follows that if $P \neq NP$, then no NP–complete problem can be solved in polynomial time, with respect to the input length, by a deterministic P system with active membranes but without membrane division, constructed in polynomial time from the input. Together, these results give a characterization of $P \neq NP$ in terms of deterministic P systems.

## 1 Introduction

In [2] a new model of computation, called *P Systems*, is introduced within the framework of *Natural Computing* (bio-inspired computing). It is based upon the notion of *membrane structure* that is used to enclose *computing cells* in order to make them independent computing units. Also, a membrane serves as a communication channel between a given cell and other cells adjacent to it. This model starts from the observation that the processes which take place in the complex structure of a living cell can be considered as *computations*.

Since these computing devices were introduced several variants have been considered. A fairly complete compendium about P systems can be found at [8]. In particular, P systems with external output are studied in [4].

The different variants of P systems found in the literature are in general generating devices. Many of them have been proved to be computationally complete: they compute all Turing computable sets of natural numbers or all recursively enumerable languages, depending on the variant considered.

The model we consider here works with symbol–objects and it has two characteristics that have seldom been considered before: we work with *decision* devices whose work is triggered by certain *input data*. The aim is to use this kind of P systems to deal with decision problems.

The main goal of this paper is to show a sufficient condition for the relation $P \neq NP$ to be verified: if there exists an NP–complete problem that cannot be

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 388–399, 2003.

solved in polynomial time, with respect to the input length, by any family of deterministic decision P systems, *constructed* in polynomial time, then $P \neq NP$.

To achieve this, we prove that every decision problem which can be solved by a deterministic Turing machine in polynomial time can also be solved by a family of deterministic decision P systems in polynomial time.

The paper is organized as follows: Section 2 briefly presents some basic concepts about P systems with external output; Section 3 introduces the new model (with symbol–objects) of *decision* P systems; Section 4 shows how to simulate deterministic Turing machines by families of such P systems; Section 5 establishes our main results about decision P systems and the $P \neq NP$ conjecture.

## 2 Multisets, Membrane Structures, Evolution Rules

A *multiset* over a set, $A$, is a mapping $m : A \to \mathbb{N}$; $m(a)$ is the number of copies of $a \in A$ in the multiset $m$. The set $\{a \in A : m(a) > 0\}$ is called the *support* of $m$ and it is denoted by $supp(m)$. A multiset, $m$, is said to be empty (resp. finite) if its support is empty (resp. finite). If $m$ is a finite multiset over $A$, we will denote it $m = \{\{a_1, \ldots, a_m\}\}$, where the elements $a_i \in supp(m)$ are possibly repeated. We write $M(A)$ for the set of all the multisets over $A$. For two multisets $m_1$, $m_2$ over $A$ we define their union by $(m_1 \cup m_2)(a) = m_1(a) + m_2(a)$, for each $a \in A$.

The set of *membrane structures*, $MS$, is defined by recursion as follows: 1. $[\,] \in MS$; 2. If $\mu_1, \ldots, \mu_n \in MS$, then $[\mu_1 \ldots \mu_n] \in MS$.

A membrane structure, $\mu$, can also be seen as a rooted tree, $\big(V(\mu), E(\mu)\big)$. Then, the nodes of this tree are called *membranes*, the root node the *skin membrane*, and the leaves *elementary membranes*. The *degree* of a membrane structure is the number of membranes in it.

The *membrane structure with environment* associated with the membrane structure, $\mu$, is $\mu^E = \big[_E \mu\big]_E$. If we consider $\mu^E$ as a rooted tree, then the root node is called the *environment* of $\mu$.

Given an alphabet, $\Gamma$, we associate with every membrane of a membrane structure a finite multiset of elements of $\Gamma$, which are called the *objects* of the membrane.

We also associate with every one of these membranes a finite set of *evolution rules*. An evolution rule over $\Gamma$ is a pair $(u, v)$, usually written $u \to v$, where $u$ is a string over $\Gamma$ and $v = v'$ or $v = v'\delta$, where $v'$ is a string over

$$\Gamma \times (\{here, out\} \cup \{in_l : l \in V(\mu)\})$$

and $\delta$ is a special symbol not in $\Gamma$. The idea behind a rule is that the objects in $u$ "evolve" into the objects in $v'$, moving or not to another membranes and possibly dissolving the original membrane.

The *length* of a rule is the number of symbols involved in the rule (for instance, the length of $u \to v$ is $|u| + |v| + 1$).

## 3 Decision P Systems

**Definition 1.** *A decision P system is a construct*

$$\Pi = (\Gamma, \Sigma, \mu_\Pi, i_\Pi, \mathcal{M}_1, \ldots, \mathcal{M}_p, (R_1, \rho_1), \ldots, (R_p, \rho_p)),$$

*where:*

- $\Sigma$ *is an alphabet, called the input alphabet.*
- $\Gamma$ *is an alphabet such that* $\Sigma \subseteq \Gamma$*; its elements are called objects; there are two distinguished objects,* $YES, NO \in \Gamma - \Sigma$.
- $\mu_\Pi$ *is a membrane structure of degree* $p$*, the membranes of which we suppose labeled from* 1 *to* $p$.
- $i_\Pi \in \{1, \ldots, p\}$ *is the input membrane of* $\Pi$.
- $\mathcal{M}_i$ *is a multiset over* $\Gamma - \Sigma$ *associated with the membrane labeled by* $i$*, for every* $i = 1, \ldots, p$.
- $R_i$ *is a finite set of evolution rules over* $\Gamma$ *associated with the membrane labeled by* $i$*, and* $\rho_i$ *is a strict partial order over* $R_i$*, for every* $i = 1, \ldots, p$.

To formalize the semantics of this model we define first what a configuration of such a P system is, and then the notion of computation.

**Definition 2.** *Let* $\Pi$ *be a decision P system with external output.*

1. *A configuration of* $\Pi$ *is a pair* $(\mu^E, M)$*, where* $\mu$ *is a membrane structure such that* $V(\mu) \subseteq V(\mu_\Pi)$ *and it has the same root than* $\mu_\Pi$*, and* $M$ *is an application from* $V(\mu^E)$ *into* $M(\Gamma)$*. For every node* $nd \in V(\mu^E)$ *we denote* $M_{nd} = M(nd)$.
2. *The initial configuration of* $\Pi$ *for the multiset* $m \in M(\Sigma)$ *is the pair* $(\mu^E, M)$*, where* $\mu = \mu_\Pi$*,* $M_E = \emptyset$*,* $M_{i_\Pi} = m \cup \mathcal{M}_{i_\Pi}$ *and* $M_j = \mathcal{M}_j$*, for every* $j \neq i_\Pi$.

The idea is that for every input multiset $m \in M(\Sigma)$, we add that multiset to the input membrane, $i_\Pi$, of the P system and then start the work of $\Pi$.

We can pass, in a non-deterministic manner, from one configuration of $\Pi$ to another by applying to its multisets the evolution rules associated with their corresponding membranes. This is done as follows: given a rule $u \to v$ of a membrane $i$, the objects in $u$ are removed from $M_i$; then, for every $(ob, out) \in v$ an object $ob$ is put into the multiset associated with the parent membrane (or the external environment if $i$ is the skin membrane); for every $(ob, here) \in v$ an object $ob$ is added to $M_i$; finally, for every $(ob, in_j) \in v$ an object $ob$ is added to $M_j$ (if $j$ is not a child membrane of $i$, then the rule cannot be applied). Finally, if $\delta \in v$, then the membrane $i$ is dissolved (if $i$ is the skin membrane, the rule cannot be applied), that is, it is removed from the membrane structure. The objects of a dissolved membrane remain in the region surrounding it, while the rules are removed. Moreover, the *priority relation* among the rules forbids the application of a rule if another one of higher priority is applied.

Given two configurations, $C$ and $C'$, of $\Pi$, we say that $C'$ is obtained from $C$ in one *transition step*, and we write $C \Rightarrow C'$, if we can pass from the first to the

second one by using the evolution rules appearing in the membrane structure of $C$ in a parallel and maximal way in each membrane, and for all the membranes at the same time.

**Definition 3.** *Let $\Pi$ be a decision P system. A computation, $\mathcal{C}$, of $\Pi$ with input $m \in M(\Sigma)$ is a sequence, possibly infinite, of configurations of $\Pi$, $C_0 \Rightarrow C_1 \Rightarrow \ldots \Rightarrow C_q$, $q \geq 0$, such that*

- *$C_0$ is the initial configuration of $\Pi$, with the multiset $m$ placed in membrane $i_\pi$.*
- *Each $C_i$ $(1 \leq i \leq q)$ is obtained from the previous configuration by one transition step.*

*We say that $\mathcal{C}$ is a halting computation of $\Pi$ if there is no rule applicable to the objects present in its last configuration. In this case, we say that $C_q$ is the halting configuration of $\mathcal{C}$.*

*We say that $\Pi$ is deterministic if for each $m \in M(\Sigma)$ there exists an unique computation with input $m$.*

The philosophy of the P systems with external output is that we cannot know what is happening inside the membrane structure, but we can only collect the information sent out from it to the environment. Thus, it is natural that the halting computations of these P systems report to the environment when they have reached their final configurations (accepting or rejecting). Furthermore, the idea behind the decision P systems is to use them as languages decision devices. These considerations lead us to the following notions.

**Definition 4.** *A deterministic decision P system, $\Pi$, is said to be* valid *when the following is verified:*

- *All computations of $\Pi$ halt.*
- *For each computation of $\Pi$ only one rule of the form $u \to v(ob, out)$, where $ob = YES$ or $ob = NO$, may be applied in the skin membrane of $\mu_\Pi$, and only in the last step of the computation.*

**Definition 5.** *Let $\Pi$ be a deterministic valid decision P system. We say that a configuration $(\mu^E, M)$ of $\Pi$ is an accepting (resp., rejecting) configuration if $YES \in M_E$ (resp., $NO \in M_E$).*

*We say that $\mathcal{C}$ is an accepting (resp., rejecting) computation of $\Pi$ if its associated halting configuration is an accepting (resp., rejecting) configuration.*

**Definition 6.** *A deterministic valid decision P system, $\Pi$, accepts (respectively, rejects) a multiset $m \in M(\Sigma)$ if the computation of $\Pi$ with input $m$ is an accepting (resp. rejecting) computation.*

We denote by $\mathcal{D}$ the class of all deterministic valid decision P systems.

## 4 Simulating Turing Machines by Decision P Systems

In what follows we are going to define what we mean by simulating a Turing machine (as a languages generating device) through a family of deterministic decision P systems. This has to be done in such a way that every solution for a decision problem given by a Turing machine provides a solution for the same problem by a decision P system. Moreover, the additional costs of the reduction from one solution to another must be polynomial in terms of the input size.

We take as a model the concept of complexity classes in membrane systems introduced by G. Păun in [1].

**Definition 7.** *We say that a deterministic Turing machine, $TM$, is simulated in polynomial time by a family of deterministic valid decision P systems $\mathbf{\Pi_{TM}} = (\Pi_{TM}(1), \Pi_{TM}(2), \ldots, \Pi_{TM}(k), \ldots)$ if:*

1. *The family $\mathbf{\Pi_{TM}}$ is $\mathcal{D}$–consistent; that is, for each $k \in \mathbf{N}^+$, $\Pi_{TM}(k)$ is a deterministic valid decision P system.*
2. *The family $\mathbf{\Pi_{TM}}$ is $TM$–uniform; that is, there exists a deterministic Turing machine, $TM'$, which constructs $\Pi_{TM}(k)$ in polynomial time starting from $k \geq 1$ (there exists a polynomial $p'(k)$ depending on $TM$ such that for each $k$, $TM'(k)$ halts in less than $p'(k)$ steps and its output is $\Pi_{TM}(k)$).*
3. *The family $\mathbf{\Pi_{TM}}$ is polynomially bounded; that is, there exists a polynomial $p(k)$, depending on $TM$, such that every computation of $\Pi_{TM}(k)$ always halts in less than $p(k)$ steps.*
4. *The family $\mathbf{\Pi_{TM}}$ is $TM$–sound; that is, the Turing machine $TM$ accepts (resp. rejects) the input string $a_{i_1} \ldots a_{i_k}$ if and only if $\Pi_{TM}(k)$ accepts (resp. rejects) $g(a_{i_1} \ldots a_{i_k})$ ($g$ is a suitable polynomial encoding of strings by multisets).*

*Note 1.* The fact that the family $\mathbf{\Pi_{TM}}$ is $\mathcal{D}$–consistent has the consequence that for each $k \geq 1$, the P system $\Pi_{TM}(k)$ has a polynomial size in the following sense: the size of the working alphabet, the number of membranes, the size of the initial multisets, and the sum of the lengths of all the rules, is bounded by $k^r$, for some constant $r$ depending on $TM$.

*Note 2.* A suitable polynomial encoding, $g$, of strings by input multisets of $\Pi_{TM}(k)$ means the following: there exists a Turing machine, $TM''$, and a polynomial $q(k)$ depending on $TM$ such that for each input data $w$ of $TM$ we have that $TM''(w)$ halts in less than $q(|w|)$ steps and its output is $g(w)$ (an input multiset of the P system $\Pi_{TM}(|w|)$).

**Theorem 1.** *Each deterministic Turing machine can be simulated in polynomial time by a family of deterministic valid decision P systems.*

*Proof.* We consider deterministic Turing machines following [6].

Suppose we have $Q_{TM} = \{q_N, q_Y, q_0, \ldots, q_n\}$, $\Gamma_{TM} = \{B, \triangleright, a_1, \ldots, a_m\}$, $\Sigma_{TM} = \{a_1, \ldots, a_p\}$, with $p \leq m$, and $\delta_{TM}(q_i, a_j) = (q_{Q(i,j)}, a_{A(i,j)}, D(i,j))$ as set of states, working alphabet, input alphabet and transition function for $TM$, respectively. We denote $a_B = B$ and $a_0 = \triangleright$.

We construct a family of deterministic decision P systems $\mathbf{\Pi_{TM}} = (\Pi_{TM}(1), \Pi_{TM}(2), \ldots, \Pi_{TM}(k), \ldots)$ which simulates $TM$ as follows: for each $k \in \mathbf{N}$, the decision P system $\Pi_{TM}(k)$ is:

- Input alphabet: $\Sigma_k = \{\langle a, i\rangle : a \in \Sigma_{TM}, 1 \leq i \leq k\}$
- Working alphabet: $\Gamma_k = \{\langle a, i\rangle : a \in \Sigma_{TM}, 0 \leq i \leq k\} \cup \{t_i : 1 \leq i \leq k\} \cup \{s_i^-, s_i^+, s_i : i \in \{T_1, T_2, F, S, 1, \ldots, 9\}\} \cup \{q_N, q_Y, h, h', YES, NO\} \cup \{q_i : 0 \leq i \leq n\} \cup \{b_i, b_i', b_i'', c_i : 0 \leq i \leq m\}$
- Membranes structure: $\mu_{\Pi} = [\,_1\,]_1$.
- Input membrane: $i_{\Pi} = 1$.
- Initial multisets: $\mathcal{M}_1 = \{\{q_0, b_0, s_{T_1}^-, s_{T_2}^-, s_F^-, s_S^-, s_1^-, \ldots, s_9^-, s_{T_1}\}\}$.
- Evolution rules: $R = R_0 \cup R_1 \cup R_2 \cup R_3 \cup R_4$, where:

- $R_0 = R_{0,1} \cup R_{0,2} \cup R_{0,3} \cup R_{0,4}$, with

$$R_{0,1} \equiv s_{T_1} s_{T_1}^- \to s_{T_1}^+ > s_{T_1}^- \to s_{T_1}^- > \begin{cases} \langle a_i, j\rangle \to \langle a_i, j\rangle t_j & (1 \leq i \leq p, 1 \leq j \leq k) \\ s_{T_1}^+ \to s_{T_1}^- s_{T_2} \end{cases}$$

$$R_{0,2} \equiv \begin{cases} s_{T_2} s_{T_2}^- \to s_{T_2}^+ > s_{T_2}^- \to s_{T_2}^- > t_1^2 s_{T_2}^+ \to s_{T_2}^- s_F > \cdots > \\ > t_k^2 s_{T_2}^+ \to s_{T_2}^- s_F > t_1 \ldots t_k s_{T_2}^+ \to s_{T_2}^- s_S > s_{T_2}^+ \to s_{T_2}^- s_F \end{cases}$$

$$R_{0,3} \equiv s_F s_F^- \to s_F^+ > s_F^- \to s_F^- > \begin{cases} s_{T_1}^- s_{T_2}^- s_S^- s_1^- \ldots s_9^- s_F^+ q_0 b_0 \to (NO, out) \\ \langle a_i, j\rangle \to \lambda \quad (1 \leq i \leq p, 1 \leq j \leq k) \\ t_j \to \lambda \quad (1 \leq j \leq k) \end{cases}$$

$$R_{0,4} \equiv \left\{ s_S s_S^- \to s_S^+ > s_S^- \to s_S^- > \left\{ \begin{array}{l} \langle a_i, j\rangle \to \langle a_i, j-1\rangle^2 \quad (1 \leq i \leq p, 1 \leq j \leq k) \\ \langle a_i, 0\rangle \to b_i \quad (1 \leq i \leq p) \end{array} \right\} \right. \\ \left. > s_S^+ \to s_S^- s_1 \right\}$$

- $R_1 = R_{1,1} \cup R_{1,2} \cup R_{1,3}$, with

$$R_{1,1} \equiv s_1 s_1^- \to s_1^+ > s_1^- \to s_1^- > \begin{cases} h \to h h' \\ b_i \to b_i b_i' \quad (0 \leq i \leq m) \\ s_1^+ \to s_1^- s_2 \end{cases}$$

$$R_{1,2} \equiv \left\{ \begin{array}{l} h' s_2 s_2^- \to s_2^+ > s_2 \to s_3 > s_2^- \to s_2^- > \\ \qquad > b_i'^2 \to b_i'' \quad (0 \leq i \leq m) > \begin{cases} b_i' \to \lambda \quad (0 \leq i \leq m) \\ b_i'' \to b_i' \quad (0 \leq i \leq m) \\ s_2^+ \to s_2^- s_2 \end{cases} \end{array} \right.$$

$$R_{1,3} \equiv s_3 s_3^- \to s_3^+ > s_3^- \to s_3^- > \begin{cases} {b'}_i^2 \to \lambda \quad (0 \leq i \leq m) \\ s_3^+ \to s_3^- s_4 \end{cases}$$

- $R_2 = R_{2,1} \cup R_{2,2}$, with

$$R_{2,1} \equiv s_4 s_4^- \to s_4^+ > s_4^- \to s_4^- > \begin{cases} h \to hh' \\ s_4^+ \to s_4^- s_5 \end{cases}$$

$$R_{2,2} \equiv s_5 s_5^- \to s_5^+ > s_5^- \to s_5^- > \begin{cases} \text{Rules for the transition} \\ \text{function} \\ s_5^+ \to s_5^- s_6 \end{cases}$$

The rules for the transition function, $\delta_{TM}$, are the following:

*Case 1:* state $q_r$, element $a_s \neq B$

| *Movement* | *Rules* | |
|---|---|---|
| *left* | $q_r b'_s h \to q_{Q(r,s)} b'_s c_{A(r,s)}$, | if $A(r,s) \neq B$ |
| | $q_r b'_s h \to q_{Q(r,s)} b'_s$, | if $A(r,s) = B$ |
| *stand* | $q_r b'_s \to q_{Q(r,s)} b'_s c_{A(r,s)}$, | if $A(r,s) \neq B$ |
| | $q_r b'_s \to q_{Q(r,s)} b'_s$, | if $A(r,s) = B$ |
| *right* | $q_r b'_s \to q_{Q(r,s)} b'_s c_{A(r,s)} h$, | if $A(r,s) \neq B$ |
| | $q_r b'_s \to q_{Q(r,s)} b'_s h$, | if $A(r,s) = B$ |

*Case 2:* state $q_r$, no element

| *Movement* | *Rules* | |
|---|---|---|
| *left* | $q_r h \to q_{Q(r,s)} c_{A(r,s)}$, | if $A(r,s) \neq B$ |
| | $q_r h \to q_{Q(r,s)}$, | if $A(r,s) = B$ |
| *stand* | $q_r \to q_{Q(r,s)} c_{A(r,s)}$, | if $A(r,s) \neq B$ |
| | $q_r \to q_{Q(r,s)}$, | if $A(r,s) = B$ |
| *right* | $q_r \to q_{Q(r,s)} c_{A(r,s)} h$, | if $A(r,s) \neq B$ |
| | $q_r \to q_{Q(r,s)} h$, | if $A(r,s) = B$ |

To avoid conflicts, every rule in case 1 has higher priority than any rule in case 2.

- $R_3 = R_{3,1} \cup R_{3,2}$, with

$$R_{3,1} \equiv h' s_6 s_6^- \to s_6^+ > s_6 \to s_7 > s_6^- \to s_6^- > \begin{cases} b'_i \to {b'}_i^2 \quad (0 \leq i \leq m) \\ c_i \to c_i^2 \quad (0 \leq i \leq m) \\ s_6^+ \to s_6^- s_6 \end{cases}$$

$$R_{3,2} \equiv s_7 s_7^- \to s_7^+ > s_7^- \to s_7^- > \begin{cases} b_i b'_i \to \lambda \quad (0 \leq i \leq m) \\ c_i \to b_i \quad (0 \leq i \leq m) \\ s_7^+ \to s_7^- s_8 \end{cases}$$

- $R_4 = R_{4,1} \cup R_{4,2}$, with

$$R_{4,1} \equiv s_8 s_8^- \to s_8^+ > s_8^- \to s_8^- > \begin{cases} q_Y \to q_Y s_9, \quad q_N \to q_N s_9 \\ q_i \to q_i s_1 \quad (0 \leq i \leq n) \\ s_8^+ \to s_8^- \end{cases}$$

$$R_{4,2} \equiv \quad s_9 s_9^- \to \lambda > s_9^- \to s_9^- > \begin{cases} s_1^- \dots s_8^- \to \lambda \\ q_Y \to (YES, out) \\ q_N \to (NO, out) \\ h \to \lambda \\ b_i \to \lambda \quad (0 \leq i \leq m) \end{cases}$$

Let us see that $\mathbf{\Pi_{TM}} = (\Pi_{TM}(1), \Pi_{TM}(2), \dots, \Pi_{TM}(k), \dots)$ is a family of deterministic valid decision P systems which simulates $TM$.

Obviously it is a family of deterministic valid decision P systems. Moreover, $\mathbf{\Pi_{TM}}$ is an *uniform family*. Indeed, let $TM$ be a deterministic Turing machine such that the set of states has size $n+2$, the working alphabet has size $m+2$ and the input alphabet has size $p$ (with $p \leq m$). The necessary resources to construct $\Pi_{TM}(k)$ are the following:

1. The size of the working alphabet $\Gamma_k$ is $p \cdot k + 4m + n + 45$; that is, in the order $\theta(k \cdot m + n)$.
2. The degree of the P system is 1.
3. The size of the initial configuration for each $x \in \Sigma_{TM}^k$ is $k + 16 \in \theta(k)$.
4. The total number of rules is in the order of
$$O(p \cdot k + n \cdot m) = O(k \cdot m + n \cdot m)$$
5. The greatest length of a rule is $16 \in O(1)$.

Let us see now that, for each $k \in \mathbb{N}$, $\Pi_{TM}(k)$ is *sound*: let $L$ be the language decided by $TM$. In order to decide if a string $a_{i_1} \dots a_{i_k} \in \Sigma_{TM}$, of length $k$, belongs to $L$, we encode it by the multiset $\{\{\langle a_{i_1}, 1\rangle, \dots, \langle a_{i_k}, k\rangle\}\}$ which is the input given to $\Pi_{TM}(k)$. The rules of this P system have been carefully chosen in such a way that its work goes through the following main stages:

1. Check that the multiset received as input codes a string of length $k$. For this, we have to verify that for each $j = 1, \dots, k$ there exists one and only one pair whose second component is equal to $j$. Otherwise, the P system halts and rejects the multiset.
2. Transform the input multiset into another multiset which encodes the string in base 2 (in order to specify the symbol written in each cell of the tape).
3. Read the element in the cell scanned by the head.
4. Compute the new element to be writen in the cell, move the head and change state (according to the transition function).
5. Erase the old element and write the new element.
6. Check if a final state is reached: if not, repeat from stage 2; if $q_Y$ is the final state reached, then accept the string; if $q_N$ is the final state reached, then reject the string.

These stages will be carried out in several small steps, each of them managed by a group of rules. To avoid rules from distinct steps being applied together, we will use $s_j^-$ as *forbidding objects* and $s_j^+$ as *permitting objects*; if $s_j^-$ is present in the P system, then rules for step $j$ *cannot* be executed; if, instead, $s_j^+$ is the object present, then rules for step $j$ *must* be executed (if possible). Of course, at

any time, for each step only the corresponding forbidding or permitting object will be present. Also, there will always exist only one permitting object in the system.

To indicate that we want to perform a step, we will use $s_j$ as *promoter objects*. When $s_j$ appears in the P system, it will transform its corresponding forbidding object $s_j^-$ into the permitting one $s_j^+$, thus allowing the rules for step $j$ to be applied. Then the permitting object will transform itself again into the forbidding one, and into a suitable promoter object.

The first two stages take care of filtering the input multisets: those which do not encode a string of size $k$ are rejected; those which do encode such strings are transformed in such a way that we have a code in base 2 of the symbols in the cells of the tape of the Turing machine. These operations are perfomed by the rules in $R_0$.

For a multiset to correctly encode a string of size $k$, it has to verify two conditions: for each $j = 1, \ldots, k$, it has to contain no more than one pair with second component equal to $j$; for each $j = 1, \ldots, k$, it has to contain at least one pair with second component equal to $j$. These two conditions are checked by rules in $R_{0,1}$ and $R_{0,2}$, using objects $t_j$ as signals for the second components of the pairs.

If the check fails, all objects in the P system are eliminated and it sends out the object NO, to reject the multiset. If the check passes, then we double $j$ times each pair of the form $\langle a, j\rangle$, changing them at the end by a $b_j$.

For a detailed description of the simulation of the running of the Turing machine over a correct multiset see [6]. We only recall here how we represent the Turing machine inside the P system.

To represent the states we will use objects $q_N, q_Y, q_0, \ldots, q_n$.

During the simulation, objects $b_0, \ldots, b_m$ will represent, in base 2, the cells which contain symbols $a_0, \ldots, a_m$, respectively (note that, at any time, the number of non-empty cells in the tapes of the Turing machine is finite); objects $b'_0, \ldots, b'_m, b''_0, \ldots, b''_m$ will be used as working copies of the previous objects; the single prime objects will also be useful to indicate the symbols read from the cells, and objects $c_0, \ldots, c_m$ will be useful to indicate the new symbols to write into them.

The cell scanned by the head, numbered from zero, will be represented by the object $h$, in base one; object $h'$ will be used, when needed, as a working copy of object $h$; it will be used as a counter.

Finally, let us see that the family $\mathbf{\Pi_{TM}}$ is *polynomially bounded*. Indeed, if $x \in \Sigma_{TM}^k$ is an input string of size $k$ of the Turing machine, then we have:

1. The check for a multiset to correctly encode a string of size $k$ needs four steps in the affirmative case and six steps in the negative one (in this last case, the P system halts).
2. The generation of the multiset encoding, in base 2, the non-empty cells in the initial configuration of $TM$ for $x$ requires $k + 3$ steps of the P system.
3. The simulation of each transition step of the Turing machine requires a cost in the order of $O(5j + 12)$, where $j$ is the cell being read by the head of $TM$.

4. Checking if a final state has been reached requires 2 steps.
5. The output after the detection of a final state requires 2 steps.

Therefore, if $TM$ accepts or rejects $x$ in $O(t(k))$ steps, then the P system $\Pi_{TM}(k)$ needs $O(k + t^2(k))$ steps to do the same. □

## 5 The Main Result

In [7], C. Zandron, C. Ferretti, and G. Mauri prove the following result.

**Theorem 2.** *If $P \neq NP$, then no NP–complete problem can be solved in polynomial time, with regard to the input length, by a deterministic P system (with active membranes but without membrane division).*

In this section we prove a kind of reciprocal result of the previous theorem through the solvability of a decision problem by a family of deterministic valid decision P systems.

Recall that a decision problem, $X$, is a pair $(E_X, f_X)$ such that $E_X$ is a language over a certain alphabet and $f_X$ is a boolean mapping over $E_X$. The elements of $E_X$ are called instances of the problem $X$. For each $k \in \mathbb{N}$ we note $E_X^k$ the language of all the instances of $X$ with size $k$.

**Definition 8.** *We say that a decision problem, $X$, is solvable in polynomial time by a family of deterministic valid decision P systems $\mathbf{\Pi_X} = (\Pi_X(k))_{k \in \mathbf{N}^+}$ if:*

1. *The family $\mathbf{\Pi_X}$ is $\mathcal{D}$–consistent; that is, for each $k \in \mathbf{N}$, $\Pi_X(k)$ is a deterministic valid decision P system.*
2. *The family $\mathbf{\Pi_X}$ is $X$–uniform; that is, there exists a Turing machine, $TM'$, and a polynomial $p'(k)$ depending on $X$ such that for each $k \in \mathbf{N}^+$, $TM'(k)$ halts in less than $p'(k)$ steps and its output is the P system $\Pi_X(k)$.*
3. *The family $\mathbf{\Pi_X}$ is polynomially bounded; that is, there exists a polynomial $p(k)$ depending on $X$ such that every computation of $\Pi_X(k)$ always halts in less than $p(k)$ steps.*
4. *The family $\mathbf{\Pi_X}$ is $X$–sound; that is, for every $a \in E_X^k$, $f_X(a) = 1$ if and only if $\Pi_X(k)$ accepts the multiset $g(a)$ ($g$ is a suitable polynomial encoding of elements of $E_X$ by input multisets of $\Pi_X(k)$).*

*Note 3.* As in Note 1, from the fact that the family $\mathbf{\Pi_X}$ is $\mathcal{D}$–consistent we infer that for each $k \geq 1$, the P system $\Pi_X(k)$ has a polynomial size, in the following sense: the size of the working alphabet, the number of membranes, the size of the initial multisets, and the sum of the lengths of all the rules, is bounded by $k^r$, for some constant $r$ depending on $X$.

*Note 4.* A suitable polynomial encoding, $g$, of elements of $E_X$ by input multisets of $\Pi_X(k)$ means the following: there exists a Turing machine, $TM''$, and a polynomial $q(k)$ depending on $X$ such that for each input data $w \in E_X$ we have that $TM''(w)$ halts in less than $q(|w|)$ steps and its output is $g(w)$ (an input multiset of the P system $\Pi_X(|w|)$.

*Note 5.* Given a Turing machine, $TM$, as a languages generating device, we consider the decision problem, $X_{TM}$, associated with $TM$, as follows: $X_{TM} = (\Sigma^*_{TM}, f_{TM})$, where $f_{TM}(w) = 1$ if and only if $TM$ accepts $w$. According with this definition we infer that a Turing machine is simulated in polynomial time by a family of deterministic valid decision P systems, $\mathbf{\Pi_{TM}}$, if and only if the associated decision problem, $X_{TM}$ is solvable in polynomial time by the family $\mathbf{\Pi_{TM}}$.

**Theorem 3.** *If there exists an NP-complete problem that cannot be solved in polynomial time, with regard to the input length, by a family of deterministic decision P systems, then* $P \neq NP$.

*Proof.* Let us suppose that $P = NP$. Then, there exists an NP–complete problem, $X$, and a deterministic Turing machine, $TM_X$, solving $X$ in polynomial time with regard to the input length (actually, all NP–complete problem verifies this property). From Theorem 1, $TM_X$ is simulated in polynomial time by a family of deterministic valid decision P systems, $\mathbf{\Pi_{TM_X}}$. Then, according with Definition 8, $\mathbf{\Pi_{TM_X}}$ solves the problem $X$ in polynomial time with regard to the input length. This leads to a contradiction. □

## 6 Final Remarks

In this paper we made clear an apparent theoretical interest of P systems without membrane division as a tool which allows us to attack the $P \neq NP$ conjecture. The search of an *adequate* NP–complete problem and the study of its solvability through such P systems will give us a direct answer to the conjecture. If the considered problem is solvable in polynomial time, then the conjecture will have an affirmative answer; otherwise, it will have a negative answer.

We think that this result provides an additional attractiveness to the research of P systems because it allows us to attack the $P \neq NP$ conjecture within this model.

## References

1. Păun, G.: *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
2. Păun, G.: Computing with Membranes, *Journal of Computer and System Sciences*, **61**(1), 2000, 108–143.
3. Păun, G.; Rozenberg, G.: A Guide to Membrane Computing, *Theoretical Computer Sciences*, **287**, 2002, 73–100.
4. Păun, G.; Rozenberg, G.; Salomaa, A.: Membrane Computing with External Output, *Fundamenta Informaticae*, **41**(3), 2000, 313–340.
5. Pérez Jiménez, M.J.; Romero-Jiménez, A.; Sancho Caparrini, F.: *Teoría de la Complejidad en modelos de computación celular con membranas*, Editorial Kronos, Sevilla, 2002.
6. Romero-Jiménez, A.; Pérez-Jiménez, M.J.: Simulating Turing Machines by P Systems with External Output, *Fundamenta Informaticae*, **49**(1-3), 2002, 273–287.

7. Zandron, C.; Ferretti, C.; Mauri, G.: Solving NP-Complete Problems Using P Systems with Active Membranes, in *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer-Verlag, London, 2000, 289–301.
8. The P Systems Web Page: `http://psystems.disco.unimib.it/`

# P Systems without Priorities Are Computationally Universal

Petr Sosík[1] and Rudolf Freund[2]

[1] Institute of Computer Science
Silesian University, Opava, Czech Republic
petr.sosik@fpf.slu.cz
[2] Department of Computer Science
Technische Universität Wien, Wien, Austria
rudi@emcc.at

**Abstract.** The "classical" model of P systems was introduced by Gheorghe Păun in 1998; this model with symbol objects was shown to be computationally universal in [8], provided that catalysts and priorities of rules are used. We now show by reduction via register machines that the priorities may be omitted from the model without loss of computational power. As a consequence, several universality results for P systems in [10] are improved.

## 1 Introduction

P systems, a symbol and string manipulating model of living cell systems, turned out to be capable to describe formally both inter-cellular and intra-cellular interactions of molecules or more complex objects, at the description level suitable from computer science point of view. Up to now, many types of P systems were presented, differing mainly in various kinds of operations inspired by aspects of behavior of living cells and their membranes.

In many cases, these models have also been shown to be computationally universal. Various proof techniques have been used for proving computational universality, the most popular one being probably the reduction of a matrix grammar with appearance checking to the given model of P systems.

There are, however, some other models of universal computers with properties similar to those of P systems. Among them, we here focus on *register machines* presented by Minsky in [7]. The machine is equipped with a fixed number of registers for storing an arbitrarily large non-negative integer. There is a program with labelled instructions, which the machine runs. The program consists of several simple instruction types like incrementation and conditional decrementation. Comparing these register machines with P systems with symbol objects, we stress that:

– both models compute in a unary system, being capable to represent numbers only by a collection of objects which are mutually undistinguishable;

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 400–409, 2003.

- both models are capable to add or remove a single object and to recognize whether a collection is empty;
- in both cases there is a fixed number of these collections that can be manipulated.

In the further parts of the paper we show that the reduction of a register machine to a P system is a powerful and elegant proof technique allowing to improve some important up-to-date universality results of P systems.

## 2 Definitions

Before proceeding to a formal description of register machines and P systems, we fix some basic notations first. For an alphabet $V$, by $V^*$ we denote the free monoid generated by $V$ under the operation of concatenation; the *empty string* is denoted by $\lambda$, and $V^* \setminus \{\lambda\}$ is denoted by $V^+$. Any subset of $V^+$ is called a *$\lambda$-free (string) language*. Moreover, by $\mathbf{N}_0$ we denote the set of non-negative integers and by $N_0^\beta RE$ we denote the family of recursively enumerable sets of $\beta$-vectors $(y_1, ..., y_\beta)$ of non-negative integers.

For more notions as well as basic results from the theory of formal languages, the reader is referred to [2] and [12].

### 2.1 Register Machines

In this subsection we briefly recall the concept of Minsky's register machine. Minsky showed (e.g., see [7]) that the universal computational power can be reached by such an abstract machine using a finite number of registers for storing arbitrarily large non-negative integers. The machine runs a program consisting of numbered instructions of several simple types. Several variants of the machine with different number of registers and different instruction sets were shown to be computationally universal (e.g., see [7] for some original definitions and [4] for the definitions we use in this paper).

An *$n$-register machine* is a construct $M = (n, P, i, h)$ where

- $n$ is the number of registers,
- $P$ is a set of labelled instructions of the form $j : (op(r), k, l)$, where $op(r)$ is an operation on register $r$ of $M$, $j, k, l$ are labels from the set $Lab(M)$ (which numbers the instructions in a one-to-one manner),
- $i$ is the initial label, and
- $h$ is the final label.

The machine is capable of the following instructions:

**(A(r),k,l)** Add one to the contents of register $r$ and proceed to instruction $k$ or to instruction $l$; in the deterministic variants usually considered in the literature we demand $k = l$.

**(S(r),k,l)** If register $r$ is not empty then subtract one from its contents and go to the instruction $k$, otherwise proceed to instruction $l$.

**HALT** Stop the machine. This additional instruction can only be assigned to the final label $h$.

In their *deterministic variant*, such $n$-register machines can be used to compute any partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$; starting with $(n_1, ..., n_\alpha) \in \mathbf{N}_0^\alpha$ in registers 1 to $\alpha$, $M$ has computed $f(n) = (r_1, ..., r_\beta)$ if it halts in the final label $h$ with registers 1 to $\beta$ containing $r_1$ to $r_\beta$. If the final label cannot be reached, $f(n)$ remains undefined.

A deterministic $n$-register machine can also analyse an input $(n_1, ..., n_\alpha) \in \mathbf{N}_0^\alpha$ in registers 1 to $\alpha$, which is recognized if the register machine finally stops by the halt instruction with all its registers being empty. If the machine does not halt, the analysis was not successful.

In their *non-deterministic variant*, $n$-register machines can compute any recursively enumerable set of non-negative integers (or of vectors of non-negative integers). Starting with all registers being empty, we consider a computation of the $n$-register machine to be successful, if it halts with the result being contained in the first ($\beta$) register(s) and with all other registers being empty.

The results proved in [4] (based on the results established in [7]) as well as in [5] and [6] immediately lead us to the following results which differ from the original results mainly by the fact that the result of a computation is stored in registers that must not be decremented:

**Proposition 1.** *For any partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$ there exists a deterministic $(\alpha + 2 + \beta)$-register machine $M$ computing $f$ in such a way that, when starting with $(n_1, ..., n_\alpha) \in \mathbf{N}_0^\alpha$ in registers 1 to $\alpha$, $M$ has computed $f(n_1, ..., n_\alpha) = (r_1, ..., r_\beta)$ if it halts in the final label $h$ with registers $\alpha + 3$ to $\alpha + 2 + \beta$ containing $r_1$ to $r_\beta$; if the final label cannot be reached, $f(n_1, ..., n_\alpha)$ remains undefined.*

**Proposition 2.** *For any recursively enumerable set $L \subseteq \mathbf{N}_0^\beta$ 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 halts with $n_i$ in registers $i$, $3 \leq i \leq \beta + 2$, and registers 1 and 2 being empty if and only if $(n_1, ..., n_\beta) \in L$.*

### 2.2 The Standard Model of P Systems

We here do not give a broad informal description of the standard type of P systems since this model has been studied in many papers and several monographs. We refer to [1], [3], [8], [9], and [10] for motivation and examples. We only note that the topology of a P system is described by a system of membranes forming

a tree structure. Each membrane region contains objects and rules applicable to these objects due to further specifications. The definition of the P system below omits some ingredients not needed in the following.

A *P system (of degree* $m, m \geq 1$*)* is a construct

$$\Pi = (V, C, \mu, w_1, \ldots, w_m, (R_1, \rho_1), \ldots, (R_m, \rho_m), i_o),$$

where:

(i) $V$ is an alphabet; its elements are called *objects*;
(ii) $C \subseteq V$ is a set of *catalysts*;
(iii) $\mu$ is a membrane structure consisting of $m$ membranes (usually labelled with $1, 2, \ldots, m$);
(iv) $w_i, 1 \leq i \leq m$, are strings over $V$ associated with the regions $1, 2, \ldots, m$ of $\mu$; they represent multisets of objects present in the regions of $\mu$ (the multiplicity of a symbol in a region is given by the number of occurrences of this symbol in the string corresponding to that region);
(v) $R_i, 1 \leq i \leq m$, are finite sets of *evolution rules* over $V$ associated with the regions $1, 2, \ldots, m$ of $\mu$, and $\rho_i$ is a partial order relation over $R_i$ (a *priority* relation); these evolution rules are of the forms $a \to v$ or $ca \to cv$, where $c$ is a catalyst, $a$ is an object from $V \setminus C$, and $v$ is a string over

$$(V \setminus C) \times \{here, out, in\};$$

(vi) $i_o$ is a number between 1 and $m$ and it specifies the *output* membrane of $\Pi$.

The membrane structure and the multisets represented by $w_i$ in $\Pi$ constitute the *initial configuration* of the system. A transition between configurations is governed by the application of the evolution rules which is done in parallel: All objects, from all membranes, which *can be* the subject of local evolution rules, as prescribed by the priority relation, *have to* evolve simultaneously.

The application of a rule $u \to v$ in a region containing a multiset $M$ results in subtracting from $M$ the multiset identified by $u$, and then in adding the multiset identified by $v$. The objects can eventually be transported through membranes due to targets *in* and *out*. We refer to [1] and [10] for further details and examples.

The system continues parallel steps until there remain no applicable rules in any region of $\Pi$; then the system halts. We consider the number of objects from $V \setminus C$ contained in the output membrane $i_o$ at the moment when the system halts as the *result* of the underlying computation of $\Pi$. The set of results of all computations possible in $\Pi$ is denoted by $N(\Pi)$. The class of all sets of $\beta$-vectors $(y_1, ..., y_\beta)$ of non-negative integers computable by P systems of the above type is denoted by $N_0^\beta OP(cat, pri)$. The notation *cat* or *pri*, respectively, is omitted when the corresponding ingredient is not used.

## 3 Universality Results

We are now ready to prove that any partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$ can be computed by a P system with only one membrane and with catalysts

only, but without priorities, and, in turn, any partial recursive set of vectors of non-negative integers can be generated by a P system with one membrane and with catalysts, but without priorities.

For a register machine $M$ with the registers $\mathbf{r}_1, \ldots, \mathbf{r}_m$, $m \geq 1$, let $P$ be the program for $M$ with $n$ instructions $i_1$, $i_2$, ..., $i_n$ computing $f$. Informally, each register $\mathbf{r}_i$ is represented by objects $o_i$, $o_i'$, playing the roles of counter elements. The value of register $\mathbf{r}_i$ at each moment corresponds to the number of symbols $o_i$ and $o_i'$ in the system.

There are also special objects $p_j$, $1 \leq j \leq n$, within the P system, playing the role of program counters (labels). The presence of the object $p_j$ starts the sequence of operations corresponding to the instruction $i_j$.

**Theorem 1.** *For each partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$ there is a P system*

$$\Pi_{\text{cat}} = (V, C, [_1]_1, w, (R, \emptyset), 1)$$

*with catalysts and with the objects $o_i \in V$ satisfying the following conditions:*

*For any arbitrary $(x_1, ..., x_\alpha) \in \mathbf{N}_0^\alpha$, denote*

$$\Pi_{\text{cat},(x_1,\ldots,x_\alpha)} = (V, C, [_1]_1, wo_1^{x_1} \ldots o_\alpha^{x_\alpha}, (R, \emptyset), 1).$$

*The system $\Pi_{\text{cat},(x_1,\ldots,x_\alpha)}$ halts if and only if $f(x_1, ..., x_\alpha)$ is defined, and if it halts, then*

$$N\left(\Pi_{\text{cat},(x_1,\ldots,x_\alpha)}\right) = \{f(x_1, ..., x_\alpha)\}.$$

*Proof.* Consider a (deterministic) register machine $M$ as described above with $m'$ registers, the last $\beta$ registers being special output registers which are never decremented. (From the result stated in Proposition 1 we know that $m' = \alpha + 2 + \beta$ is sufficient). Now let $P$ be a program which computes the function $f$ and let $m = m' - \beta$.

The input values $x_1, ..., x_\alpha$ are expected to be in $\mathbf{r}_1$ to $\mathbf{r}_\alpha$ and the output values from $f(x_1, ..., x_\alpha)$ are expected to be in $\mathbf{r}_{m+1}$ to $\mathbf{r}_{m'}$. Moreover, without loss of generality, we may assume that at the beginning of a computation all the registers except eventually $\mathbf{r}_1$ to $\mathbf{r}_\alpha$ contain zero.

We construct the P system

$$\Pi_{\text{cat}} = (V, C, [_1]_1, w, (R, \emptyset), 1),$$

where

$$\begin{aligned} V &= \{r, r', \#\} \cup \{o_i, o_i', c_i, c_i' \mid 0 \leq i \leq m\} \\ &\quad \cup \{o_k \mid m < k \leq m'\} \cup \{p_j, p_j', p_j'' \mid 1 \leq j \leq n\}, \\ C &= \{c_i, c_i' \mid 0 \leq i \leq m\}, \\ w &= r'^m p_1'' o_0 o_0' c_0 c_0' c_1 c_1' \ldots c_m c_m'. \end{aligned}$$

Then for an arbitrary $(x_1, ..., x_\alpha) \in \mathbf{N}_0^\alpha$ the axiom of the corresponding system $\Pi_{\text{cat},(x_1,\ldots,x_\alpha)}$ is

$$r'^m p_1'' o_0 o_0' c_0 c_0' c_1 c_1' \ldots c_m c_m' o_1^{x_1} \ldots o_\alpha^{x_\alpha}.$$

The contents of the register $\mathbf{r}_i$, $1 \leq i \leq m$, is represented by the sum of the number of symbols $o_i$ and $o'_i$, and actions on this register are guarded by the pair of catalysts $c_i, c'_i$. Moreover, we use something like an additional register $\mathbf{r}_0$; it contains the number two represented by one symbol $o_0$ and one symbol $o'_0$ until the halting instruction $i_n$ is simulated (and only in this case $o_0$ and $o'_0$ may disappear) and it is controlled by the pair of catalysts $c_0, c'_0$.

The set of rules $R$ depends on the instructions of $P$ and is constructed as follows:

(i) Each instruction of the program $P$ is simulated by two steps of $\Pi_{\text{cat}}$. We have to guarantee that some catalysts ($c'_i$) can be applied with the "computing" rules in (ii) and (iii) only in the even steps and some others ($c_i$) only in the odd steps. Hence, we use the auxiliary objects $r, r' \in V$ and the trap object # as well as the rules

$$r \to \#, \quad r' \to \#, \quad \# \to \#, \quad c_i r \to c_i r', \quad c'_i r' \to c'_i r$$

with $1 \leq i \leq m$ in $R$. This ensures that the rules $c_i r \to c_i r'$ and $c'_i r' \to c'_i r$ must be used at each even and odd step, respectively. Otherwise the trap object appears and the system never halts. Hence, the catalysts $c_i$ and $c'_i$, respectively, can be used with other rules only in the remaining steps.
Moreover for each $i$, $0 \leq i \leq m$, the set $R$ contains the rules

$$c_i o_i \to c_i o'_i, \quad c'_i o'_i \to c'_i o_i, \quad o'_i \to \#, \quad c'_i o_i \to c'_i.$$

If the contents of register $\mathbf{r}_i$ is non-zero, then after each odd step the object $o'_i$ appears. This object is used in (iii) for decrementation by using the rule $c'_i o_i \to c'_i$ for $1 \leq i \leq m$.
Finally, there is the rule

$$p''_1 \to p_1$$

in $R$, producing the symbol $p_1$ corresponding to the first instruction of the program $P$. Each instruction except *Halt* then is performed in two or four subsequent steps of the system $\Pi_{\text{cat},(x_1,\ldots,x_k)}$, starting in an even step.

(ii) For each $j$, $1 \leq j < n$, such that $i_j = (A(a), k, k)$ (remember that $M$ is a deterministic register machine) for some $a, k$ with $1 \leq a \leq m'$ and $1 \leq k \leq n$, there are the rules

$$p_j \to p'_k o_a, \qquad p'_k \to p_k$$

in $R$. Their meaning is obvious – the number of symbols $o_a$ is incremented.

(iii) For each $j$, $1 \leq j < n$, such that $i_j = (S(a), k, l)$ for some $a, k, l$ with $1 \leq a \leq m$ and $1 \leq k, l \leq n$, we add to $R$ the rules

$$c_a p_j \to c_a p''_k, \quad p''_k \to p'_k, \quad p'_k \to p_k, \quad c'_a p_j \to c'_a p'_l, \quad p'_l \to p_l,$$

which are used in the following way: Assume that the object $p_j$ appears in the system at the beginning of an even step.

(a) If the contents of the register $\mathbf{r}_a$ is zero, then there is no object $o'_a$ within the system. Then the rules $c'_a p_j \to c'_a p'_l$ and $p'_l \to p_l$ are subsequently applied. Hence, the instruction $i_l$ is simulated next.

(b) If $\mathbf{r}_a$ is non-zero, then there is one object $o'_a$ within the system. The rule $c'_a p_j \to c'_a p'_l$ cannot be applied since the trap object # would appear by the enforced application of the rule $o'_a \to \#$; the rule $c'_a o'_a \to c'_a o_a$ must be used instead.
Therefore, in the following odd step the rule $c_a p_j \to c_a p''_k$ can be applied. (If the rule $c_a o_a \to c_a o'_a$ were used instead, the object $p_j$ would remain unchanged and the situation would be repeated until the rule $c_a p_j \to c_a p''_k$ is chosen).
In the following even step, there is neither $o'_a$ nor $p_j$ (we now have $p''_k$ instead) present in the system and therefore the rule $c'_a o_a \to c'_a$ must be used, decrementing the number of symbols $o_a$ by 1; moreover, from $p''_k$ we obtain $p'_k$ by applying $p''_k \to p'_k$. In the next step, we finally obtain $p_k$ using $p'_k \to p_k$; hence, the instruction $i_k$ is simulated next.

Any other behavior of the system leads to the appearance of the trap object # within the system.

(iv) To stop the computation after finishing the simulation of the program $P$, the last instruction $i_n = Halt$ is represented by the rule

$$c_0 p_n \to c_0.$$

When the halting instruction $i_n$ is reached, this rule $c_0 p_n \to c_0$ can be applied, which allows to set the "register" $\mathbf{r}_0$ to zero, e.g., by the following sequence of (partial) reactions in the P system:

$$c_0 c'_0 o_0 o'_0 p_n \Longrightarrow c_0 c'_0 o_0 o_0 \Longrightarrow c_0 c'_0 o'_0 \Longrightarrow c_0 c'_0 o_0 \Longrightarrow c_0 c'_0.$$

Now the catalyst $c'_0$ is "free for use" and can be applied within the following rules:

$$c'_0 r \to c'_0, \qquad c'_0 r' \to c'_0.$$

In this way all the objects $r$ and $r'$ can be removed from the system. Then also all the objects $o_i, o'_i$, $1 \le i \le m$ are removed thanks to the rules $c'_i o'_i \to c'_i o_i$, $c'_i o_i \to c'_i$ (listed already in (i)), and the system can halt. Then, except for the catalysts $c_i, c'_i$, $0 \le i \le m$, only the multiset of symbols $o_{m+1}$ to $o_{m'}$ that represents the result of the computation is still present in the system.

Any other behavior of the system than that described above leads to producing the trap object #, hence, the system never halts.

In such a way, the system $\Pi$ simulates a sequence of instructions of the program $P$ beginning with the first instruction $i_1$ and halting after having reached the instruction $i_n = Halt$. It follows from the description given above that after each performing of an instruction the number of objects $o_i, o'_i$ equals the contents of register $\mathbf{r}_i$, $1 \le i \le m'$. Hence, after performing the instruction $Halt$

and halting the system, the number of symbols $o_{m+1}$ to $o_{m+\beta}$ equals the output of the program $P$. The only other objects remaining within the system are the $2(m+1)$ catalysts; according to the result about register machines stated in Proposition 1, $m = \alpha + 2$ and therefore $2(\alpha+3)$ catalysts ($\alpha+3$ pairs of corresponding catalysts) are enough. □

**Corollary 1.** *$N_0^\beta OP_d(cat) = N_0^\beta RE$ for every $d \geq 1$.*

*Proof.* We only have to prove the relation $N_0^\beta RE \subseteq N_0^\beta OP_1(cat)$. In the same way as in the proof of Theorem 1 the P system $\Pi_{\text{cat}}$ was constructed in order to simulate the (deterministic) register machine from Proposition 1, we now construct a P system $\Pi'_{\text{cat}}$ which simulates the non-deterministic register machine $M$ with program $P$ from Proposition 2 and in that way non-deterministically generates a representation of any vector from the given language $L$ in $N_0^\beta RE$ by the corresponding numbers of symbols $o_3$ to $o_{2+\beta}$. Hence, let us define

$$\Pi'_{\text{cat}} = (V, C, [_1]_1, w', (R', \emptyset), 1),$$

where $w' = r'^2 p''_1 o_0 o'_0 c_0 c'_0 c_1 c'_1 c_2 c'_2$ and $R'$ is constructed in a similar way as in the preceding proof, except that now in the non-deterministic case we have Add-instructions of the form $i_j = (A(a), k, l)$ for some $a, k, l$ with $1 \leq a \leq \beta + 2$ and $1 \leq k, l \leq n$ which are simulated by the rules

$$p_j \to p'_k o_a,\ p_j \to p'_l o_a, \qquad p'_k \to p_k,\ p'_l \to p_l$$

in $R'$. Obviously, $N(\Pi'_{\text{cat}}) = L$. By the given construction, we only need $(2+\beta)$ pairs of catalysts. □

Obviously, the results obtained so far are optimal with respect to the number of membranes in the P systems constructed in the proofs of Theorem 1 and Corollary 1. On the other hand, it remains an open question which is the minimal number of catalysts we really need for these P systems; observe that without priorities as well as without catalysts, too, we can only generate regular sets. Hence, the following results are also optimal with respect to the number of membranes where we consider the standard definition of P systems which says that in the output membrane no other symbols than the terminal symbols should appear (e.g., see [10]):

**Corollary 2.** *For each partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$ there is a P system*

$$\Pi_{\text{cat}} = (V, C, [_1[_2]_2]_1, w, \lambda, (R, \emptyset), (\emptyset, \emptyset), 2)$$

*with catalysts and with the objects $o_i \in V$ satisfying the following conditions:*

*For any arbitrary $(x_1, ..., x_\alpha) \in \mathbf{N}_0^\alpha$, denote*

$$\Pi_{\text{cat},(x_1,...,x_\alpha)} = (V, C, [_1[_2]_2]_1, wo_1^{x_1} ... o_\alpha^{x_\alpha}, \lambda, (R, \emptyset), (\emptyset, \emptyset), 2).$$

*The system* $\Pi_{\mathrm{cat},(x_1,\ldots,x_\alpha)}$ *halts if and only if* $f(x_1,\ldots,x_\alpha)$ *is defined, and if it halts, then in the skin membrane only the catalysts remain and in the output membrane* 2 *only terminal symbols* $o_{\alpha+3}$ *to* $o_{\alpha+2+\beta}$ *appear in such a way that*

$$N\left(\Pi_{\mathrm{cat},(x_1,\ldots,x_\alpha)}\right) = \{f(x_1,\ldots,x_\alpha)\}.$$

*Proof.* We only have to change some of the rules simulating Add-instructions of the form $i_j = (A(a), k, k)$ for some $a, k$ with $\alpha + 3 \leq a \leq \alpha + 2 + \beta$ and $1 \leq k \leq n$, i.e., we now take

$$p_j \to p'_k(o_a, in), \qquad p'_k \to p_k$$

in $R$, i.e., symbols representing the contents of output registers are immediately sent into the output membrane 2. □

**Corollary 3.** *For each set* $L$ *in* $N_0^\beta RE$ *there is a P system*

$$\Pi'_{\mathrm{cat}} = \left(V, C, [_1[_2]_2]_1, r'^2 p''_1 o_0 o'_0 c_0 c'_0 c_1 c'_1 c_2 c'_2, \lambda, (R', \emptyset), (\emptyset, \emptyset), 2\right)$$

*with catalysts such that* $N(\Pi'_{\mathrm{cat}}) = L$ *and in a halting computation of* $\Pi'_{\mathrm{cat}}$ *the skin membrane contains only the catalysts whereas the output membrane only contains symbols* $o_3$ *to* $o_{2+\beta}$.

*Proof.* As in the preceding proof we only have to change some of the rules simulating Add-instructions of the form $i_j = (A(a), k, l)$ for some $a, k, l$ with $3 \leq a \leq \beta + 2$ and $1 \leq k, l \leq n$, i.e., we now take

$$p_j \to p'_k(o_a, in),\ p_j \to p'_l(o_a, in), \qquad p'_k \to p_k,\ p'_l \to p_l$$

in $R'$. □

## 4 Conclusion

The technique used in Theorem 1 can also be interpreted as follows: It is known that a rather similar result was presented in [11] using so-called *bi-stable catalysts*. These catalysts oscillate between two states; the catalyst is switched to the opposite state when used. Notice that in the proof of Theorem 1 we use pairs of catalysts such that those of the first type can be used in odd steps only, and those of the second type can be used in even steps only, and the use of the catalysts of the second type is conditionally determined by the use of the catalysts of the first type (otherwise a trap object is produced). Hence, their behavior is quite similar to those of bi-stable catalysts.

The number of catalysts used in the P systems constructed in the proofs of this paper can be seen as a complexity measure for these systems. It remains an interesting open question whether the bounds for the number of catalysts needed in the proofs above (which at least are optimal with respect to the number of membranes) can be improved.

Finally we should like to point out that there are some more theorems in [10] (as well as in some older articles and monographs) which are improved by the results obtained in this paper; we refer to [13] for more details.

**Acknowledgements**

Petr Sosík's research was supported by the Grant Agency of Czech Republic, grant No. 201/02/P079.

## References

1. Calude, C.S., Păun, Gh.: Computing with Cells and Atoms. Taylor & Francis, London (2001)
2. Dassow, J., Păun, Gh.: Regulated Rewriting in Formal Language Theory. Springer-Verlag, Berlin (1989)
3. Dassow, J., Păun, Gh.: On the Power of Membrane Computing. Journal of Universal Computer Science **5**, 2 (1999) 33–49 (`http://www.iicm.edu/jucs`)
4. Freund, R., Oswald, M.: GP Systems with Forbidding Context. Fundamenta Informaticae **49**, 1-3 (2002) 81–102
5. Freund, R., Păun, Gh.: On the Number of Non-terminals in Graph-controlled, Programmed, and Matrix Grammars. In: Margenstern, M., Rogozhin, Y. (eds.): Proc. Conf. Universal Machines and Computations, Chişinău (2001). Springer-Verlag, Berlin (2001)
6. Freund, R., Păun, Gh.: From Regulated Rewriting to Computing with Membranes: Collapsing Hierarchies; *to appear in TCS*
7. Minsky, M.L.: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, New Jersey (1967)
8. Păun, Gh.: Computing with Membranes. Journal of Computer and System Sciences **61,** 1 (2000) 108–143, and TUCS Research Report 208 (1998) (`http://www.tucs.fi`)
9. Păun, Gh.: Computing with Membranes: An Introduction. Bulletin EATCS **67** (1999) 139–152
10. Păun, Gh.: Membrane Computing: An Introduction. Springer-Verlag, Berlin (2002)
11. Păun, Gh., Yu, S.: On Synchronization in P Systems. Fundamenta Informaticae **38**, 4 (1999) 397–410
12. Salomaa, A., Rozenberg, G. (eds.): Handbook of Formal Languages. Springer-Verlag, Berlin (1997)
13. Sosík, P.: The Power of Catalysts and Priorities in Membrane Systems; *to appear in Grammars*

# The Architecture of Living Structures – A Possible Basis for Molecular Computing

Roxana Vasilco, Alina Popescu, Ruxandra Chiurtu, and Dan Dascalu

National R&D Institute of Microtechnology
Bucharest, Romania
http://www.imt.ro
roxanav@imt.ro

**Abstract.** The paper recalls basic biological facts about living structures, as useful supports for molecular computing. Particular emphasis is made on membranes and proteins, whose structural and conformational dynamics might be considered as generating a computational environment in living systems.

The fundamental challenge of biology is to explain the behavior, function, organization, and evolution of living structures. This generic question is followed by more specific problems to be solved, such as:

- The way(s) by which organic molecules, mainly nucleic acids, proteins, and carbohydrates, coordinate and integrate in order to guarantee cell functioning and reproduction.
- The way(s) cells preserve their individuality and specificity, despite genotypic and environmental stresses.
- The way(s) organisms adapt to environmental changes by evolution and transformation at the molecular level.
- The way(s) to match diversity, adaptability, and evolution of cells.

The very recent achievements of molecular biology may represent a good reason to hope to solve a number of problems and to offer a practical approach to fundamentals of life itself. The most striking (and convenient) feature of the protein molecules, as basic elements, is the self-assembling dynamics they possess, with very simple governing rules to be easily transferred into any computational system. The prime and ultimate expression of "molecular computational efficiency" is life itself. Any organism, whether simple or complex, functions as a result of a tremendous series of interactions between huge numbers of different components made of protein molecules. Every molecular component displays fast and ultra fast dynamic behaviors, including phase transitions and adaptative restructurations.

The solutions to the questions mentioned in the first paragraph rely on the deep and clear knowledge of relationship existing between the different levels on which life is based:

- the relationship between proteins sequence, structure and function,

G. Păun et al. (Eds.): WMC-CdeA 2002, LNCS 2597, pp. 410–421, 2003.

– the relationship between proteins assembled into networks,
– the relationship between networks existing and acting inside cells as a whole,
– the relationship between cells in a population or an organism,
– the relationship between organisms in populations.

The sequential basis of life is thus requiring the use of computational tools in order to get complex and general information on living structures. Obviously, such an approach is meant to gather biologists, but also chemists, physicists, mathematicians, computer scientists, and engineers, to extract, analyze and interpret data leading to an overall understanding of life. The main limitation imposed by the biological condition itself, which involves too many non-quantifying interrelations and chronic lack of specific measuring systems and tools, may thus be outridden and a valuable modelling/computing tool gained.

That nature applies common assembly rules is implied by the recurrence – at scales from the nanoscopic to the macroscopic – of certain patterns, such as spirals, pentagons, and triangulated forms. These patterns appear in very different structures, from highly regular crystals (as it is the case with diamonds) to relatively irregular proteins (such as the enzymes) and, in living entities, from viruses to humans. Basically, both organic and inorganic matter are made of the same building blocks: atoms of carbon, hydrogen, oxygen, nitrogen, and phosphorus. The difference is made by how atoms are arranged in the 3D space. This phenomenon, in which components join together to form larger, stable structures with new properties that could not have been predicted from the characteristics of their individual parts, is known as *self-assembly* and is observed at every scale in nature.

Self-assembly is the coordinated action of independent entities under distributed (i.e., non-central) control to produce a larger structure or to achieve a desired group effect. Instances of self-assembly occur in biology (e.g., embryology and morphogenesis) and in chemistry (e.g., the formation of more loosely bound supramolecular structures from groups of molecules).

In the living systems, molecules self-assemble into functional macromolecules, macromolecules self-assemble into cellular components – the organelles, which self-assemble into cells, which self-assemble into tissues, which self-assemble into organs, and the final result is a body organized hierarchically as tiers of *systems within systems*. This drastically expressed hierarchical control cells display have its correspondent into the hierarchically organized *regions* of P systems, where *evolution rules* (sometimes with permitting or forbidding conditions) act on the *objects* placed in regions. The boundary between subsystems is represented by *membranes* for both biological objects (cells) and P systems.

**Membranes** represent sheet-like molecular assemblies, 6-10 nm thick, separating different compartments of a given living entity, as well as the living entity from its environment:

– *Plasma membrane* separates the cell from its extracellular environment.
– *Intracellular membranes* isolate different parts of the cell, providing functional compartments (endoplasmic reticulum and Golgi complex, involved in cell secretion, nucleus, cell).

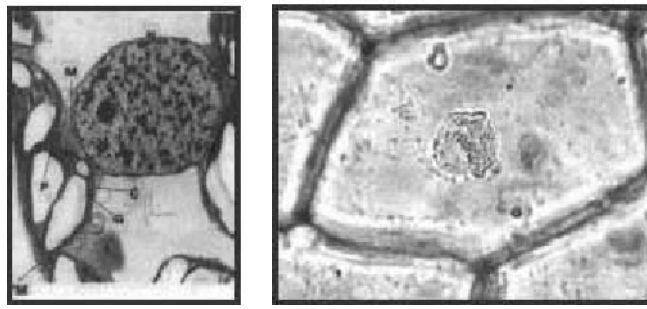


**Fig. 1.** Membrane of animal cell (left) and of plant cell (right)

**Fig. 2.** Cellular and intracellular membranes

The membranes, both plasmatic and intracellular, fairly correspond to the *membranes* of the P systems and have the same role: to define a space, the *region* equivalent for the P systems.

Both in biological and P systems, membranes undergo a division process following very similar division rules, in order to generate new *cells/P systems* and/or inner membranes describing new *spaces/regions*.

**The role biological membranes** play is complex. They act as:

- Selective and responsive barriers between discrete masses of cytoplasm and their environments. This function depends on selective channels and pumps (selectivity) and on specific receptors and transducing proteins (response to signals).
- Maintaining the homeostatic conditions inside the cell, conducive to the biochemical reactions sustaining life.

**The chemical composition** is similar in all living forms:

- Molecular species: lipids and proteins non-covalently associated.
- *Lipids* form a double layer of amphipathic (amphiphylic) species.

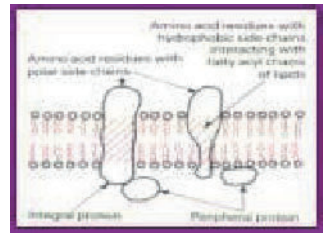


**Fig. 3.** The interaction of integral and peripheral membrane proteins with the lipid bilayer

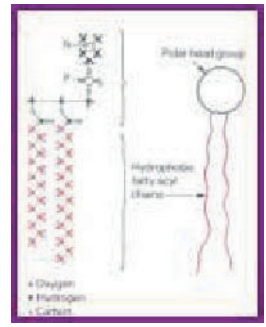


**Fig. 4.** Amphipathic lipid molecules

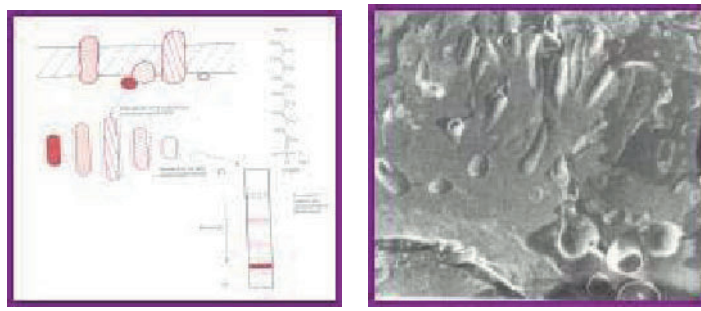


**Fig. 5.** Denaturation and solubilization of proteins (left) and Electronomicrograph of placental membranes (right)

- *Proteins* function as *transport proteins* (carrying substances in/out), *receptor proteins* (transmitting signals), *anchor proteins* (attaching the structural proteins inside/outside the cell). *Peripheral proteins* are superficially bound by weak ionic interactions (easily removed with aqueous solutions); *integral proteins* are tightly bound, being removed by detergents.
- *Carbohydrates* covalently bound either to lipid (*glycolipids*) or to proteins (*glycoproteins*); they are especially abundant in the eukaryotic plasma mem-

**Fig. 6.** Liposomes formed by sonication

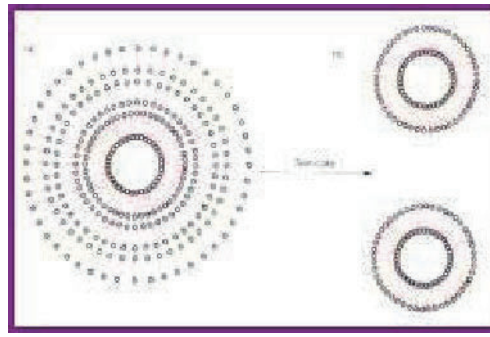


**Fig. 7.** Polar lipids and single lipid bilayer vesicles

brane, absent from intracellular membranes, and also missing from prokaryotic membranes.

By dispersion in water of phospholipidic mixtures, because of hydrophobic interactions between fatty chains, one obtains multilayered phospholipidic vesicles, called *liposomes*.

Moreover, by liposome sonication, because of electrostatic and H-bonding interactions, one obtains one-bilayer vesicles.

**The organization of cell membranes** follows a pattern called *fluid mosaic*, formed by the lipidic bilayer, with absorbed or wholly/partially embedded proteins in the membrane core. One can observe several chemical interactions between the hydrophobic regions of lipids and proteins:

– at the membrane surface – the hydrophilic region of amphiphylic molecules is solvated in the aqueous environment;

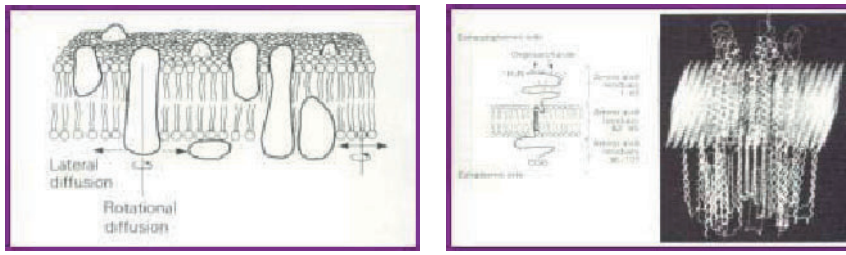



**Fig. 8.** Fluid mosaic model of membrane (left) and Protein-lipid bilayer association (right)

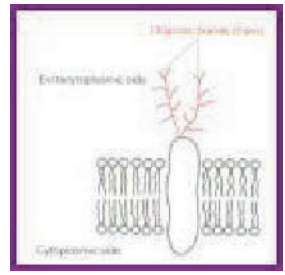


**Fig. 9.** Membrane asymmetry and boundary layers

- moving from one side of the bilayer to the other ("flip-flop") generating *gel-like* and *fluid-like* states, maintaining membranes fluidity appropriate to their environment; as a result, biological membranes maintain an *asymmetric distribution* of, mainly, lipidic components;
- between the sides of the bilayer.

  Membrane proteins also show an asymmetric distribution. Integral membrane proteins are surrounded with a "boundary layer" ("microdomain") showing a different physical state to that of the bulk, influencing the biological activity of the membrane (transport, specific recognition, etc). In general, asymmetry induces thermodynamically unfavorable states.

Every chemical species that membranes contain is equivalent to the *objects* one is operating in the P systems; there are also considered as operable *objects* the structures (organelles) that intracellular membranes are forming inside cells.

**Cell junctions** represent specialized regions of the plasma membrane joining cells into tissues with structural and functional integrity.

- Plant cells junctions, also called *plasmadesmata*, have a similar role to gap junctions, but a different structure, being cytoplasm limited by plasma membrane.
- Animal cells junctions:
  - *desmosomes* (zonulae adherens) hold cells together;

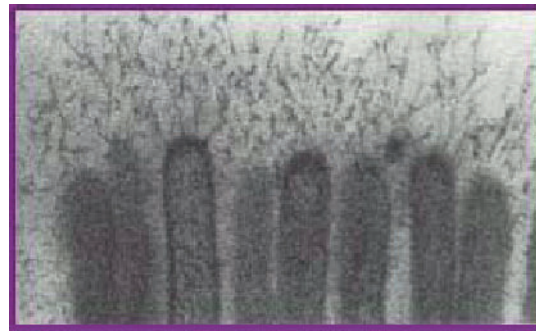


**Fig. 10.** Electronomicrograph of glycocalyx

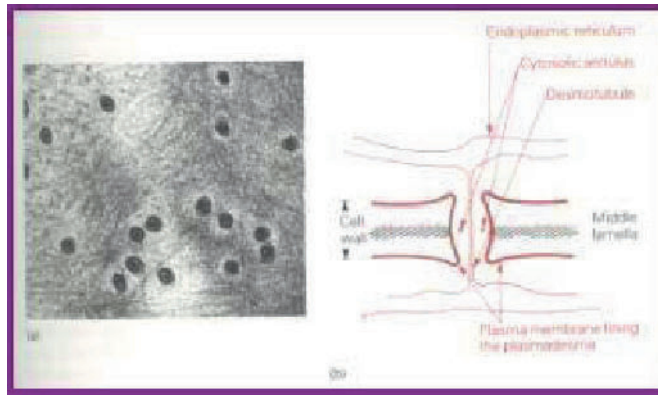



**Fig. 11.** Plasmadesmata in the plant cell wall

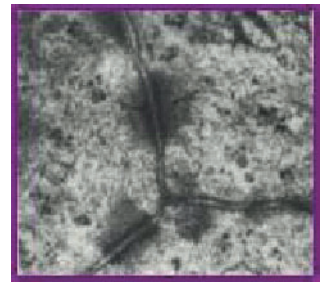


**Fig. 12.** Electronomicrograph of a desmosome

- *spot desmosomes* act mechanically like rivets, through filamentous structures (30nm);
- *hemi desmosomes* link the cell membrane to extracellular connective material (basal lamina), anchoring the cells to the matrix;

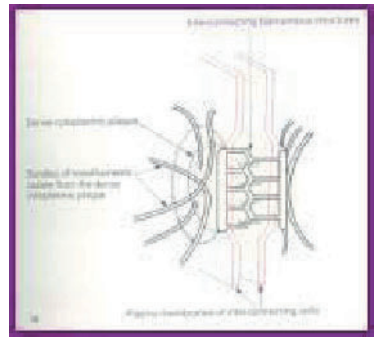


**Fig. 13.** Spot desmosomes

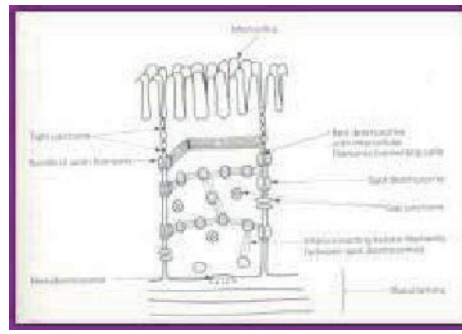


**Fig. 14.** Different types of cell junction in the small intestine epithelial cells

- *belt desmosomes* are loosely continuos bands between interconnecting cells;
- *tight junctions* (zonulae occludens) – hold cells together and seal the space between them for the molecules circulation;
- *gap junctions* form channels of communications for ions/small molecules passage (up to $M_r$ 1200, representing mainly metabolic substrates as building blocks, and secondary messengers – cAMP, $Ca^{2+}$). They permit electrical coupling for specialized cells (heart, smooth muscle), allowing the free flow of $Ca^{2+}$ necessary to maintain contractions.

**Membrane dynamics** is vital for the membrane functions. It is based on defined movements of proteins and lipids forming transient and asymmetric structures.

– *Endocytosis* is the active process of taking materials from the cell environment. It is a rapid process (seconds to minutes), involving the recycling of the membrane. Here are several particular forms of endocytosis:
  - *Pinocytosis* – the formation of membranal vesicles which trap extracellular fluid containing dissolved nutrients and transport it into the cytoplasm (e.g., macrophages).

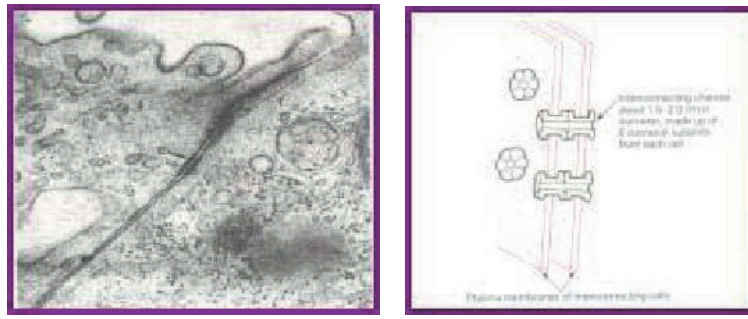


**Fig. 15.** Electronomicrograph of a tight junction (left) and Gap junction and belt desmosome

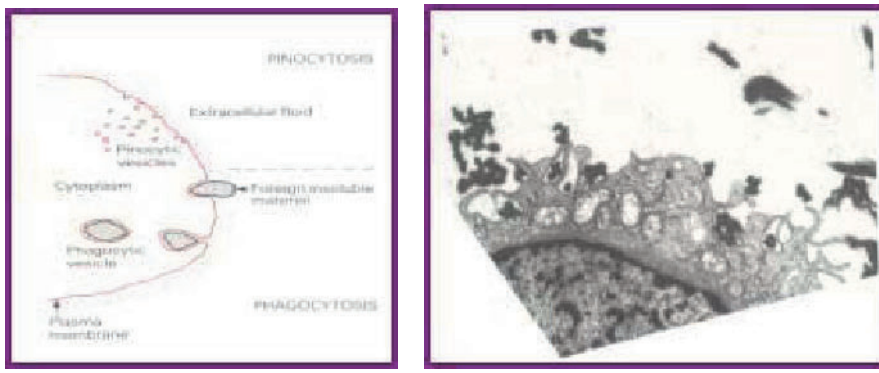



**Fig. 16.** Pynocytosis and phagocytosis (left) and Electronomicrograph of fibrin hagocytosis (right)

- *Phagocytosis* – the formation of phagocytic vacuoles to ingest large insoluble particles; digestion is performed by fused lysosomes, which deliver into the cytosol the nutrients (e.g., protozoa, polymorphonuclear leukocytes and phagocytes).
- *Receptor-mediated endocytosis* – the endocytosis process entailing binding of molecules to specific cell surface receptors (*endocytic pits*) before internalization.

– *Exocytosis* is the fusion of vesicles from the interior of the cell with the plasma membrane, to expel their content (hormones, neurotransmitters, etc.).
– *Cell signalling and cell recognition* is the dynamic function of the cell membrane allowing the *message* (e.g., hormones) *transmission* from cell to cell across the membrane by the association/dissociation of the membranal proteins or *intercellular recognition* (self/foreign), by means of specific molecules (glicoproteins, glycolipids).
– *Membrane transport* is a dynamic function meant to preserve cell homeostasis (conducing environment with regard to biochemical reactions sustaining life,

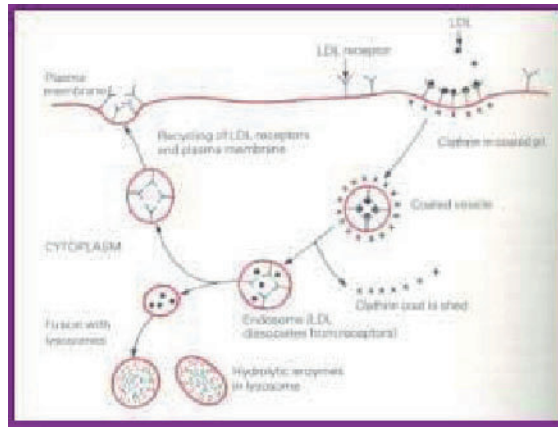



**Fig. 17.** Receptor-mediated endocytosis

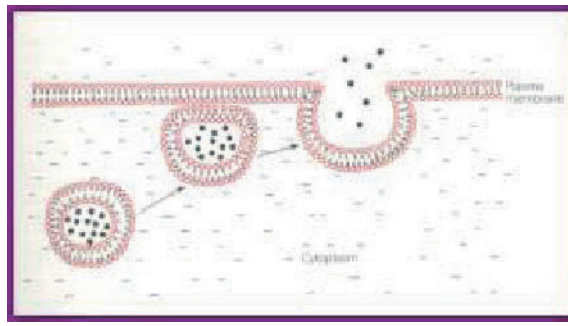



**Fig. 18.** Exocytosis

*p*H and ionic environment, acquirement of fuel molecules and excretion of toxic and secretory materials), realized due to *selective permeability*. It is of two main types, *passive* and *active*.

- The *passive diffusion* is the free movement of molecules across the membrane down a concentration gradient.
  The passive diffusion also proceeds according to *lattice theory*: diffusion may occur by molecular movement into a vacancy in the lattice structure of the lipid bilayer.
- *Facilitated* (*passive*) and *active* transport, for polar and ionic compounds, is the transport done with the aid of specific carrier molecules (proteins) embedded in the lipid bilayer; this is a highly specific process, faster than diffusion, maximal at the saturation point.
  This type of transport can be of two main types:
  - *facilitated diffusion* – for ions/molecules crossing the membrane down the electrochemical/concentration gradient, until reaching the equilibrium; it does not use metabolic energy to be performed;

**Fig. 19.** Simple diffusion through the cell membrane

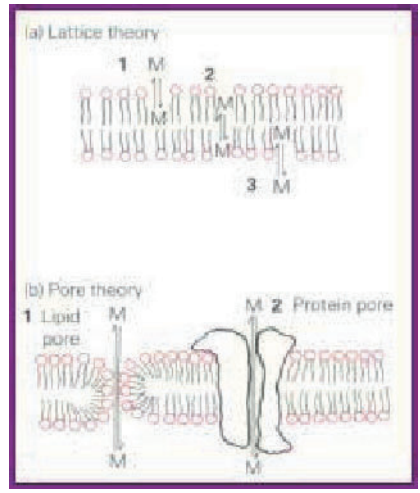



**Fig. 20.** Passive diffusion of molecules through biological membranes

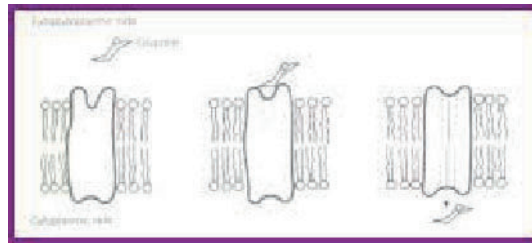


**Fig. 21.** Facilitated diffusion of molecules through biological membranes

∗ *active transport* – performed *against* the electrochemical concentration gradient, using metabolic energy.
Molecules transport across membranes may be coupled:

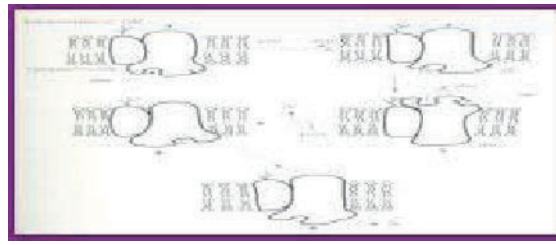


**Fig. 22.** Active transport of molecules through biological membranes

- *synergistically*, when both ions/molecules are transported in the same direction; the process is also called *symport*, and it has been already considered in P systems in the form of symport rules;
- *antagonistically*, when one ion/molecule is transported inside the cell and the other outside the cell; the process is also called *antiport*, and captured in P systems by antiport rules.

The facts we presented above suggest a strong similarity between biological and computational models, and we believe that this similarity is able to suggest/ground new computational tools, beneficial for developing both P systems theoretical apparatus and for investigating real structures and process patterns of living systems.

## References

1. J. Bailey, Mathematical Modeling and Analysis in Biochemical Engineering: Past Accomplishments and Future Opportunities, *Biotechnol. Progr.*, 14 (1998), pp. 8–20.
2. R. Freund, M. Oswald, GP Systems with Forbidding Contexts, *Fundamenta Informaticae*, 49, 1-3 (2002), pp. 81–102.
3. C.P. Marijuan, Bioinformation: Untangling the Networks of Life, *BioSystems*, 64 (2002), pp. 111–118.
4. C. Martin-Vide, A. Păun, Gh. Păun, G. Rozenberg, Membrane Systems with Coupled Transport: Universality and Normal Forms, *Fundamenta Informaticae*, 49, 1-3 (2002), pp. 1–15.
5. Gh. Păun, From Cells to Computers: Computing with Membranes (P Systems), *BioSystems*, 59 (2001), pp. 139–158.
6. G.M. Whitesides, Self-Assembling Materials, *Scientific American*, Sept. 1995, pp. 146–149.
7. G.M. Whitesides, J.P. Mathias, C.P. Seto, Molecular Self-Assembly and Nanochemistry: A Chemical Strategy for the Synthesis of Nanostructures, *Science*, 29 (Nov. 1991), pp. 1312–1318.

# Author Index