Ninth Workshop on Membrane Computing (WMC9)

Edinburgh, July 28 – July 31, 2008

Pierluigi Frisco, David W. Corne, Gheorghe Păun
Editors

Ninth Workshop on Membrane Computing (WMC9)

Edinburgh, July 28 – July 31, 2008

Pierluigi Frisco, David W. Corne, Gheorghe Păun
Editors

Workshop organised by the
School of Mathematical and Computer Sciences,
Heriot-Watt University

Edinburgh, UK, 2008

Proceedings of the 9th International Workshop on Membrane Computing (WMC9)

Techinical report HW-MACS-TR-0061 at:
School of Mathematical and Computer Sciences,
Heriot-Watt University,
Edinburgh, UK, July 2008.

Edited by: Pierluigi Frisco, David W. Corne, Gheorghe Păun, 2008

Published in July 2008

Preface

This volume contains the papers presented at the **Ninth Workshop on Membrane Computing, WMC9**, which took place in Edinburgh, UK, from July 28 to July 31, 2008. The first three workshops on Membrane Computing were organized in Curtea de Argeş, Romania – they took place in August 2000 (with the proceedings published in *Lecture Notes in Computer Science*, volume 2235), in August 2001 (with a selection of papers published as a special issue of *Fundamenta Informaticae*, volume 49, numbers 1–3, 2002), and in August 2002 (with the proceedings published in *Lecture Notes in Computer Science*, volume 2597). The next five workshops were organized in Tarragona, Spain, in July 2003, in Milan, Italy, in June 2004, in Vienna, Austria, in July 2005, in Leiden, The Netherlands, in July 2006, and in June 2007, with the proceedings published as volumes 2933, 3365, 3850, 4361, and 4860, respectively, of *Lecture Notes in Computer Science*.

The 2008 edition of WMC was organized at Heriot-Watt Univerity, by the School of Mathematical and Computer Sciences, under the auspices of the European Molecular Computing Consortium (EMCC) and IEEE Computational Intelligence Society Emergent Technologies Technical Committee Molecular Computing Task Force. In this workshop, several of the invited speakers were from outside the Membrane Computing community. This initiative was meant to allow the bioinformatics and systems biology communities to learn about and appreciate membrane systems as modeling platforms, while at the same time the membrane system community benefits by learning more about the challenges present in bioinformatics and systems biology to which membrane systems could be usefully applied.

The 7 invited speakers were: Daniela Besozzi (Milan, Italy), David Gilbert (Glasgow, UK), Paulien Hogeweg (Utrecht, The Netherlands), Markus Kirkilionis (Warwick, UK), Grzegorz Rozenberg (Leiden, The Netherlands), Francisco-José Romero-Campero (Nottingham, UK), Stephen Wolfram (Champaign - IL - USA). Full papers associated with the invited talks or only extended abstract are included in the present volume.

The volume also contains the 26 accepted papers. Each of them was subject of three or four referee reports. The program committee consisted of Artiom Alhazov (Turku, Finland, and Chişinău, Moldova), David Corne (Edinburgh, UK) – Co-chair, Pilar De La Torre (Durham, USA), George Eleftherakis (Thessaloniki, Greece), Miguel-Angel Gutiérrez-Naranjo (Sevilla, Spain), Oscar H. Ibarra (Santa Barbara, CA, USA), Fairouz Kamareddine (Edinburgh, UK), Lila Kari (London, Canada), Alica Kelemenova (Opava, Czech Republic), Jetty Kleijn (Leiden, The Netherlands), Natalio Krasnogor (Nottingham, UK), Van Nguyen (Adelaide, Australia), Linqiang Pan (Wuhan, China), Gheorghe Păun (Bucharest, Romania) – Chair, José Maria Sempere (Valencia, Spain), Gyorgy Vaszil (Budapest, Hungary), Sergey Verlan (Paris, France), Claudio Zandron

(Milano, Italy). All papers were also read by Pierluigi Frisco and Gheorghe Păun.

During the workshop two prizes: one for the *best paper* and another for *important contributions to Membrane Computing* were awarded.

The Organizing committee consisted of Pierluigi Frisco – Chair, David Corne – Co-Chair, and Elizabeth Bain Andrew – Secretary.

The invited papers and a selection of regular papers, improved according to the discussions held in Edinburgh and additionally refereed, will be published in a special issue of *Lecture Notes in Computer Science*.

Details about Membrane Computing can be found at *The P Sytems Webpage*: `http://ppage.psystems.eu` and its mirror `http://bmc.hust.edu.cn/psystems`. The workshop web site, mainly designed by David K. W. Li as his final year project, is `http://macs.hw.ac.uk/wmc9`. The logo of the workshop has been created by Anna Stoutjesdijk.

The workshop was sponsored by: the School of Mathematical and Computer Sciences at Heriot-Watt University, the Engineering and Physical Sciences Research Council (EPSRC), the International Journal on Natural Computing, Oxford University Press, and the Scottish Bioinformatics Forum. Sponsors are listed in no particular order.

The editors warmly thank all who contribute to making WMC9 a success, the organizing and programme committee, the invited speakers, the authors of the papers, the lecturers, the reviewers, and all the participants.

Pierluigi Frisco
David Corne
Gheorghe Păun
Editors

Contents

From computing to modelling with membrane systems: a critical view

Daniela Besozzi

Università degli Studi di Milano, Dipartimento di Informatica e Comunicazione,
Via Comelico 39, 20135 Milano, Italy
besozzi@dico.unimi.it

Extended abstract

Membrane systems were introduced in 1998 as distributed, parallel and nondeterministic computing models, inspired by the compartmentalized structure of eukaryotic cells and by the flow of solutes and biochemical reactions therein [20]. These cellular elements are formally represented in the standard definition of membrane systems, where multisets of objects, distributed inside separate regions, can be modified and communicated between adjacent compartments. In particular, the architecture of the cell is represented by a set of hierarchically embedded regions, each one delimited by a surrounding boundary (called membrane), and all contained inside an external main region (called the skin membrane). Inside each region, a multiset of objects and a set of evolution rules can be placed. Objects represent the formal counterpart of the molecular species floating inside cellular compartments (i.e. ions, proteins, etc.), and they are described by means of symbols or strings over a given alphabet. Evolution rules represent the formal counterpart of chemical reactions, and are given in the form of rewriting rules, which can operate on the objects (by modifying or moving them between adjacent regions), as well as on the compartmentalized structure (by dissolving, creating or dividing membranes).

With these basic components, a membrane system can perform computations, in the following way. Starting from an initial configuration, which is defined by the multiset of objects initially placed inside the compartmentalized structure, and by the sets of evolution rules, the system evolves according to an established strategy that guides the application of rules. Usually, a unique clock is assumed to beat the time steps for the whole system, in such a way that all regions proceed in a synchronized fashion (that is, the application of rules is carried out simultaneously inside all regions). In the basic class of membrane systems, evolution rules are applied in a nondeterministic and maximally parallel manner (we remark here that a rewriting rule is applicable when all the objects that appear in its left-hand side are available, in the current time step, inside the region where the rule is placed). The maximal parallelism of rule application simply means that every rule that is applicable inside a region has to be applied in that region. Stated otherwise, the maximal parallelism assures that all the objects that *could* be modified (rewritten, deleted or communicated to an adjacent region) by some rule,

are actually modified by that rule – in this sense, the parallelism of rules application also determines the maximal consumption of all (modifiable) objects. However, it might be the case that several rules compete for the same objects, and there are no enough copies of such objects to guarantee that all rules will be actually applied. In this situation, the nondeterminism comes into play: the applied rules are randomly chosen among the set of applicable rules, inside each region. As a consequence, by nondeterministically choosing different sets of rules, different evolutions of the system can be generated. So doing, starting from a fixed initial setting, a tree of computations is obtained. A computation with a membrane system is thus defined as a sequence of transitions between successive configurations, each (possibly) differing from the previous one with respect to the type or number of objects that occur in the regions, and to the regions structure possibly modified by the rules. The result of halting computations – that is, finite sequences of transitions that end in a configuration where no rule is applicable further on – is usually read in the form of a set of objects, collected during the computation in a specified region (or expelled through the skin membrane). Due to the nondeterministic nature of the system, the outputs of the system are given by all possible halting evolutions which can be reached from the initial configuration.

In ten years of research from the seminal paper by Gheorghe Păun, several classes of membrane systems have been defined by taking inspiration from different aspects of living cells (e.g., membrane charge, symport and antiport-based communication through membranes, catalytic objects, etc.), differentiated types of cells (e.g., neurons) or multicellular tissues. The computing power and efficiency of these classes have been extensively investigated via standard approaches in the area of formal languages, grammars and complexity thoeries.

Since it is not within the scope of this work to go into more depth with respect to the computing aspects of membrane systems, but rather to focus on their adoption as a modelling paradigm for biological systems, we just skip any further detail about computational issues for the time. We refer indeed the reader to [21] and to the up-to-date bibliography of membrane systems that can be found at the web address `http://ppage.psystems.eu`.

In the very last years, by taking advantage of the biologically inspired aspects of the underlying structure and formalism, membrane systems have also been considered for the modelling of biological systems, and for the investigation of their dynamical properties by means of properly defined simulators. Several applications have reported the potentialities of membrane systems for the description of natural systems at different degrees of complexity. Just to mention a few, these applications range from chemical and cellular processes, where molecule interactions [25, 22, 29] or the functioning of specific cell components [1, 6] are modelled, to broader natural systems where intercellular communication mechanisms [2, 32, 22] or interspecies dynamics in ecological systems [4, 5, 14] are considered.

Indeed, any model of a biological or chemical system mimics a physical reality by adopting a certain level of abstraction, by assuming hypothesis, and by considering some established principles for the description of the dynamical evolution of the sys-

tem. As for other standard modelling paradigms (see, e.g., [17,34]), different approaches have been considered within the field of membrane systems: in the following we briefly recall only the stochastic-based strategies that drive the system evolution (that is, the application of rules and the handling of objects and regions). We refer to [8, 9], and references therein, for the development of a deterministic-based strategy. As a preliminary step, we mention here some of the advantages and limits of membrane systems in their application to biological systems, and we defer to a forthcoming extension of this work for a thorough discussion of these description-related and evolution-related issues.

The most attractive features of membrane systems can be devised in some peculiar aspects: the distributed membrane structure, which allows to delimit distinct spatial places (where different rules can take place and operate onto local objects); the rewriting form of rules, which provides a good understandability of the description of the system; the possibility to communicate objects among regions, which grants the flow of information from a local level to the global level of the system; the parallelism at the level of regions, which gives the strong capability to keep track of the global functioning of the system, and so on. On the other side, some aspects of membrane systems – albeit being powerful features from a computational perspective – are not adequate for a proper description of many biological systems. For instance, the maximal parallelism at the level of rules or of objects consumption is surely not lifelike; the nondeterministic selection of rules indeed homogenizes the physical world where, actually, the events are not so uniformly accidental; the merely topological placement of regions cannot always catch the distribution of space, since also physical dimensions or the spatial coordinates with respect to a reference system matter, and so on.

Some possible solutions have been proposed in membrane systems models to cope with these limits. Concerning the description-related aspect, for instance, in [4] it was outlined how the topology of membrane structure is not enough to capture the "geographical" distribution of fragmented habitats where metacommunity populations live (the topological membrane structure was there transformed into a weighted graph-like structure, where node attributes were also used to describe the region dimensions), while in [7] it was proposed a parametric bidimensional skeleton to consider the distribution and movement of (trans)membrane protein populations over a membrane surface.

Another matter is concerned with the evolution-related facet. In this case, finding the most appropriate way to generate and simulate the correct dynamics of a natural system is generally far from being an easy question. In membrane systems applications, the main features that have been disputed about this problem are the maximal parallel rule application and the nondeterminism, as well as the associated matter of time. Several strategies have been adopted: in [24, 25], for instance, the evolution strategy consists in assigning to each rule a probability value – combinatorially defined according to the current multiset of objects and to left-hand side of the rule – which dynamically change from step to step. Probabilities are then used to assign objects to rules and, so doing, the selection of the rules that will be actually applied becomes mitigated with respect to the purely randomized manner of membrane systems. All regions then evolve in parallel, and get synchronized for the communication of objects at the end of each constant-time step. Yet, the consumption of objects remains maximal with this strategy. To solve this

unwanted drawback, in [4] specific rewriting rules, called mute rules, were introduced to the aim of arbitrarily reducing the maximal number of modified objects. Another strategy have been considered in [22], where the authors propose an extension of Gillespie's stochastic algorithm – a simulation procedure for well mixed, single-volumed reaction systems [15] – in order to deal with compartmentalized systems. Here, the probability associated to each rule is used to evaluate the rule's "waiting time", and then only those rules that have the same minimal waiting times (if more than one exists) with respect to all other rules are applied, but only one rule inside each region is allowed to take place. In this case, steps do not have a constant time unit, but lasts as much as the waiting time of the applied rule. Obviously, with this strategy, rules belonging to different regions that have distinct waiting times cannot be applied in parallel, otherwise it would not be possible both to define a unique time line for the whole system, and to correctly handle the communication of objects between regions. In this sense, the evolution of the system resembles more a sequential rather than parallel evolution mode, thus losing a global view on the system. This problem does not arise, on the contrary, when using the strategy proposed in [12] (see also [3] for a comprehensive description of this approach), which extends the stochastic τ-leaping algorithm given in [10]. The functioning of this system, called τ-DPP, provides that all regions proceed simultaneously, more than one rule can be applied inside each region, the communication of objects can be handled in a straightforward way, and there exists a global time stream for the whole system. Therefore, the accuracy and efficiency of τ-leaping algorithm is integrated with, and takes advantage of, the most powerful features of membrane systems. Finally, yet another stochastic strategy has been introduced in [11] where, by closely resembling Gillespie's algorithm, the membrane structure and the evolution rules are transformed in such a way that the whole system reduces to a single region, where just one rule at a time can be applied.

The majority of applications of membrane systems for biological systems concerned modelling aspects and simulation approaches, but relatively few research has been dedicated so far to develop theories and tools to perform the *analysis* of the system, or to test the effectiveness and goodness of the proposed model. We sketch hereby two approaches that already appeared in this direction, and then we point out two other issues that are still missing in the membrane systems area, but which are surely of fundamental importance to advocate that this new modelling framework is strong enough as other modelling paradigms.

As a general procedure, in order to investigate the dynamics of a modelled system, one set of initial conditions is fixed at a time, and the evolution of the system is then simulated. This process can then be repeated for any other choice of initial setting – especially for testing several conditions that can intervene on the system behaviour – which seem to be plausible or interesting from a biological or a computational point of view. The whole study can thus be time consuming, and the exhaustive sampling of too many conditions might become impractical. Moreover, when stochastic simulation frameworks are used, yet another difficulty arises: when looking for dynamical properties, it is not immediate to partition the system's phase space in homogeneous subspaces

which exhibits the same behavior. The first approach to overcome these drawbacks has been developed in [23]: it is a simulation tool that simultaneously generates the evolutions (inside single regions of a membrane system) for an entire set of initial conditions, hence allowing to quickly predict the behavior of the region under investigation. The tool is based on the construction of a grid, whose vertices correspond to different initial multisets of objects that can be placed inside a fixed region, which is then used to define a vector field over the phase space of that region. The vector field represents all the single-time step evolutions of the grid multisets, and can be exploited to characterize the stochastic nature of the system by identifying dynamical points (stable or unstable points) or the local behavior of the region (quasi-periodic or periodic orbits).

On the other side, in some cases performing simulations of a system is not suitable, for instance when a high computational burden is required, or when we are interested in revealing dynamical properties of the system without having an appropriate knowledge on the initial conditions that might generate those dynamics. In these cases, if it is plausible to assume that most of the information of the system can be stored in a stoichiometric matrix, then the system can be represented by means of an equivalent mathematical (linear) system. So doing, it is possible to exploit well established theories to extract, from the stoichiometric matrix, the information about the topology of the system's phase space. This approach has been adopted in [19] where, in the context of Markov chains theory, the phase space of (sequential) membrane systems can be partitioned into sets of configuration states, which allow to recognize dynamical patterns without performing simulations. In particular, a set of minimal communicating classes (or cycles) that "generate" the phase space is derived, and their mutual reachability is studied.

Despite the existence of these two approaches, a thorough treatment able to cope with the analysis of dynamical properties of generic membrane systems (with many regions, different communication mechanisms, stochastic evolution strategies, etc.) is still to be fully developed. Besides, what is still missing in membrane systems are operable and effective tools which can handle the lack of quantitative data, and the presence of uncertainties in the current knowledge of biological systems. It is well known that in theoretical or simulation-based studies of biological systems, several factors – such as species concentrations or molecule copy numbers, binding constants, transcription and translation rates, etc. – represent a quantitative, indispensable information for a proper investigation of the system dynamics. Most of the times, the experimental values of these factors are not available or else ambiguous, since carrying out their measurements *in vivo* or *in vitro*, at the microscopic levels, can be tangling or impossible [27]. For instance, the cellular components involved in transcription and translation mechanisms cannot be isolated and studied separately from the cell, and many other small-scale processes are accessible only through *in situ* fluorescence methods (that is, imaging methods of proteins tagged with fluorescent markers). The lack of these information, or the inaccuracy about the available data, have a direct effect on the computational study of biological systems: they result in the challenging problem of assigning biologically plausible values to all the variables defined within a model.

In some cases, the values of some parameters of a system can be estimated from *in*

vitro experiments at the macroscopic level (e.g., by fitting the dynamics derived through equations of mass-action law against the concentration curves that result from these experiments), or by assuming more or less strict analogies with other processes or organisms for which more data and knowledge are available. In general, anyway, modellists have to handle the presence of uncertainties at various levels, which are not limited only to the numerical values associated to molecular species and to reaction kinetics, but can also pertain to the structure of the system (i.e., to reveal which components are the most significant, and how they do interact each other). Having to deal with this general scarcity of knowledge, it results very hard to debate for the "validity" of a model built for a biological system, and indeed it would be more appropriate to propose the model as a corroboration (or a falsification) of the built-in description of the reality that it provides.

For this purpose, two parallel and complementary approaches can be undertaken. On the one hand, several optimization techniques can be used to "calibrate" the model, that is, to find out the unknown parameter values which allow to reproduce the expected biological dynamics in the best possible way. The parameter estimation methods usually attack this calibration problem by minimizing a cost function (i.e., a distance measure) which defines how good is the fitting of the predicted values with respect to the experimentally measured values of the same factors, or with respect to the expected biological behaviour of the system (we refer the interested reader to [27, 28, 33, 18] for some applications in Systems Biology of parameter estimation methods). On the other hand, sensitivity analysis techniques (see [30, 31] and references therein) can help in understanding how much the uncertainty in the model outcome is determined by the uncertainties, or by the variations, of the model factors (components, reactions and respective parameters). Moreover, the analysis of sensitivities of the model output can also reveal which input factors bring about the most striking effects on the system behaviour, and can thus be assumed to be good control points of the system dynamics. Traditionally, sensitivity analysis has been diffusely applied to deterministic continuous models, by means of (derivative-based) local or global methods, though theories and tools for parametric sensitivity of discrete stochastic systems have recently been defined [26, 16]. Taking advantage of these usable theories, it would be surely convenient to develop similar analysis methodologies within the field of membrane systems, especially when they are used for the investigation of biological systems.

Bibliography

[1] I.I. Ardelean, D. Besozzi, M.H. Garzon, G. Mauri, S. Roy, P system models for mechanosensitive channels, *Applications of Membrane Computing* (G. Ciobanu, G. Păun, M.J. Pérez-Jiménez Eds.), Springer–Verlag, Berlin, 43–81, 2005.

[2] F. Bernardini, M. Gheorghe, N. Krasnogor, R.C. Muniyandi, M.J. Pérez-Jiménez, F.J. Romero-Campero, On P systems as a modelling tool for biological systems, *Membrane Computing, International Workshop, WMC6*(R. Freund et al. Eds.), LNCS 3850, 114–133, 2006.

[3] D. Besozzi, P. Cazzaniga, D. Pescini, G. Mauri, A multivolume approach to stochastic modelling with membrane systems, submitted.

[4] D. Besozzi, P. Cazzaniga, D. Pescini, G. Mauri, Modelling metapopulations with stochastic membrane systems, *BioSystems*, 91, 3, 499-514, 2008.

[5] D. Besozzi, P. Cazzaniga, D. Pescini, G. Mauri, Seasonal variance in P system models for metapopulations, *Progress in Natural Science*, 17, 4, 392–400, 2007.

[6] D. Besozzi, G. Ciobanu, A P system description of the sodium–potassium pump. *Proc. of 5th Membrane Computing International Workshop (WMC04)* (G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa Eds.), LNCS 3365, 210-223, 2005.

[7] D. Besozzi, G. Rozenberg. Formalizing spherical membrane structures and membrane proteins populations, *Membrane Computing, 7th International Workshop, WMC 2006* (H.J. Hoogeboom, Gh. Păun, G. Rozenberg, A. Salomaa Eds.), Leiden, LNCS 4361, 18–41, 2006.

[8] L. Bianco, *Membrane Models of Biological Systems*. PhD Thesis, University of Verona, 2007.

[9] L. Bianco, Psim: a computational platform for metabolic P systems, *Proceedings of the 8th Workshop on Membrane Computing (WMC8)* (G. Eleftherakis, P. Kefalas, G. Păun Eds.), 1–20, 2007.

[10] Y. Cao, D.T. Gillespie, L.R. Petzold, Efficient step size selection for the tau-leaping simulation method, *Journ. Chem. Phys.*, 124:044109, 2006.

[11] M. Cavaliere, S. Sedwards, Modelling cellular processes using membrane systems with peripheral and integral proteins, *Fourth Intern. Conference on Computational Methods in Systems Biology (CMSB2006)*, LNBI 4210, 108–126, 2006.

[12] P. Cazzaniga, D. Pescini, D. Besozzi, G. Mauri, Tau leaping stochastic simulation method in P systems. *Membrane Computing. 7th International Workshop, WMC 2006* (H.J. Hoogeboom, G. Păun, G. Rozenberg, A. Salomaa Eds.), LNCS 4361, 298–313, 2006.

[13] S. Cheruku, A. Păun, F.J. Romero-Campero, M.J. Pérez-Jiménez, O.H. Ibarra, Simulating FAS-induced apoptosis by using P systems, *Progress in Natural Science*, 17, 4, 424–431, 2007.

[14] F. Fontana, V. Manca, Predator-prey dynamics in P systems rules by metabolic algorithm, *BioSystems*, 91, 3, 545–557, 2008.

[15] D.T. Gillespie, Exact stochastic simulation of coupled chemical reactions, *Journ. Phys. Chem.*, 81:2340–2361, 1977.

[16] R. Gunawan, Y. Cao, L. Petzold, F.J. Doyle III, Sensitivity analysis of discrete stochastic systems, *Biophys. J.*, 88, 2530–2540, 2005.

[17] T.C. Meng, S. Somani, P. Dhar, Modeling and simulation of biological systems with stochasticity, *In Silico Biology*, 4:0024, 2004.

[18] C.G. Moles, P. Mendes, J.R. Banga, Parameter estimation in biochemical pathways: a comparison of global optimization methods, Genome Research, 13, 2467–2474, 2003.

[19] M. Muskulus, D. Besozzi, R. Brijder, P. Cazzaniga, S. Houweling, D. Pescini, G. Rozenberg, Cycles and communicating classes in membrane systems and molecular dynamics, *Theoretical Computer Science*, 372 (2-3), 242–266, 2007.

[20] G. Păun, Computing with membranes, *Journal of Computer and System Sciences*,

61(1), 108–143, 2000 (see also *Turku Center for Computer Science*-TUCS Report No 208, 1998).

[21] G. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.

[22] M.J. Pérez-Jiménez, F.J. Romero-Campero, P systems, a new computational modelling tool for Systems Biology, Transactions on Computational Systems Biology VI (C. Priami, G. Plotkin Eds.), LNBI 4220, 176–197, 2006.

[23] D. Pescini, D. Besozzi, G. Mauri, Investigating local evolutions in dynamical probabilistic P systems, *Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'05)*, IEEE Computer Press, 440–447, 2005.

[24] D. Pescini, D. Besozzi, G. Mauri, C. Zandron, Dynamical probabilistic P systems. *International Journal of Foundations of Computer Science*, 17, 1, 183–204, 2006.

[25] D. Pescini, D. Besozzi, C. Zandron, G. Mauri, Analysis and simulation of dynamics in probabilistic P systems, *DNA Computing, 11th International Workshop on DNA Computing, DNA11* (N. Pierce, A. Carbone, eds.), LNCS 3892, 236–247, 2006.

[26] S. Plyasunov, A.P. Arkin, Efficient stochastic sensitivity analysis of discrete event systems, *J. Comput. Phys.*, 221, 724–738, 2007.

[27] S. Reinker, R.M. Altman, J. Timmer, Parameter estimation in stochastic biochemical reactions, *Systems Biology, IEE Proceedings*, 153, 4, 168–178, 2006.

[28] M. Rodriguez-Fernandez, P. Mendes, J.R. Banga, A hybrid approach for efficient and robust parameter estimation in biochemical pathways, *BioSystems*, 83, 248–265, 2006.

[29] F.J. Romero-Campero, M.J. Pérez-Jiménez, Modelling gene expression control using P systems: the Lac operon, a case study, *BioSystems*, 91, 3, 438–457, 2008.

[30] A. Saltelli, M. Ratto, S. Tarantola, F. Campolongo, sensitivity analysis for chemical models, *Chem. Rev.*, 105, 2811–2827, 2005.

[31] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, S. Tarantola, *Global Sensitivity Analysis: The Primer*, Wiley-Interscience, 2008.

[32] G. Terrazas, N. Krasnogor, M. Gheorghe, F. Bernardini, S. Diggle, M. Cámara, An environment aware P-system model of quorum sensing, *CiE 2005* (S.B. Cooper, B. Löwe, L. Torenvliet Eds.), LNCS 3526, 479–485, 2005.

[33] J. Tomshine, Y.N. Kaznessis, Optimization of a stochastically simulated gene network model via simulated annealing, *Biophysical Journal*, 91, 3196–3205, 2006.

[34] T.E. Turner, S. Schnell, K. Burrage, Stochastic approaches for modelling in vivo reactions, *Computational Biology and Chemistry*, 28, 165–178, 2004.

Modelling and analysing the dynamic behaviour of biochemical networks

David Gilbert

University of Glasgow, Bioinformatics Research Centre,
A310, Joseph Black Building, Glasgow G12 8QQ, UK
drg@brc.dcs.gla.ac.uk

Modelling intracellular biochemical networks presents particular challenges connected with the need to reconcile the difficulty of obtaining experimental data with the requirements of providing information at a level of detail which will be useful for biochemists. Such models should be able to both explain and predict behaviour, and I will illustrate this with research undertaken in Glasgow in the area of intracellular signalling pathways. Moreover modelling is a key component in the emerging discipline of Synthetic Biology in which the activity of design as well as analysis is of prime importance. I will discuss a framework for modelling and analysing biochemical networks which unifies qualitative and quantitative approaches, and show how it has been applied in the context of a Synthetic Biology project. I will also describe a very fast model checker that has been developed at Glasgow which can be applied to biochemical systems and discuss how it can be used to drive the design of such systems.

Multilevel modeling of morphogenesis

Paulien Hogeweg

Utrecht University, Theoretical Biology and Bioinformatics Group,
Padualaan 8, 3584CH Utrecht, The Netherlands
P.Hogeweg@uu.nl

In order to study the growth and development of cellular systems one needs a formalism in which on can combine the biophysical properties of cells with the modulation of these properties by gene regulatory processes. I will argue that the multisclale CA formalism now known as the Cellular Potts Model (CPM) provides a simple yet basically sound representation of a biological cell, which can be interfaced with gene regulatory processes. It represents a cell as highly deformable object which takes its shape from internal and external forces acting upon it. I will demonstrate how within this formalism complex large scale morphodynamic processes can result from local regulation of cell, and in particular membrane, properties. I will explain morphogenetic mechanisms which tend to evolve in such systems.

1 Introduction

Biological development, in particular morphogenesis, is an pre-eminent example of a process which bridges mutiple levels of organization.

For modeling these processes we need a multilevel modeling formalism which allows us to incorporate the following premisses.

- Target morhogenesis ss (not only pattern formation)
- Use cells as basic unit (growth, division, movement, ...)
- Cell is NOT point, bead, homunculus, but deformable highly viscous objects
- Genes act through cells 'with a dynamics of their own"

Moreover, although biological systems are highly complex it should allow is to formulate models which are "simple enough but not too simple" cf Einstein).

For these requirements a simple but basically correct representation of a cell is essential.

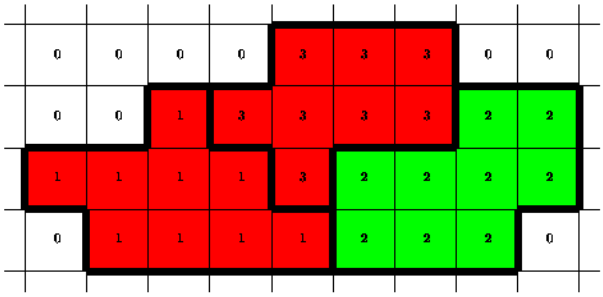


Fig. 3.1 representation of biological cells in CPM

2 Some material properties of biological cells

Properties of cells as material objects which distinguishes them from often used simplified representations are:

- Biological cells are highly dissipative systems:
  Viscosity dominates inertia
- forces acting on cells can best be described in Aristotelian regime **(F $\tilde{-}$ v)**
- resistance to shape-changes intrinsic to cell **(Yield)**
- very fast redistribution of intracellular pressure through
  osmotic pressure balance due to water flow across membrane (*exp. shown in Paramecium by Iwamoto et al 2005)*
- **Compressibility** under mechanical force *exp shown by e.g. Tricky et al 2006: they measure a Poisson ratio of .36 in chondrocytes. Volume variations upto 20% have been observed by Iwamoto et al 2005*
- **'Spontaneous' membrane fluctuations driven by cytoskeleton**
  when inhibited adhesion driven cell sorting is reduced (Mombach et al 1995)
- **Cortical tension**, and regulation thereof can modify cell shape (see review Lecuit and Lenne, 2007)

3 Modeling the generic properties of biological cells

A simple model which incorporates these properties is the so called "Cellular Potts model (CPM)" (Glazier & Graner 1993, Graner and Glazier 1992) It is a 2 scale cellular automata-like model. The model formulation involves two intrinsic scales, the micro scale on which the dynamics takes place (the CA scale) and the scale of the represented biological cell(s), whose (dynamic) properties impact on the micro-scale dynamics. This is realized as follows:

- The cells are mapped to the cellular automaton as a set of contiguous grid cells with identical state (see fig 1)

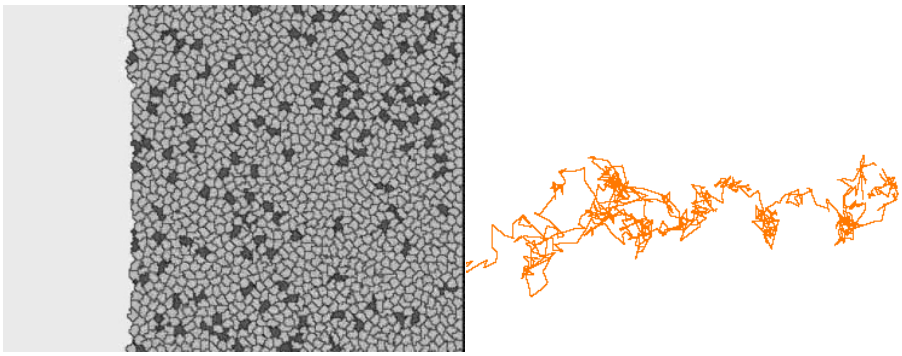


Fig. 4.2 size differences may lead to movement "against the flow", (cf Käfer et al 2006)

- Cells have an actual volume v and target volume V (in number of pixels), possibly a membrane-size m and M, a type τ (and .....)
- Between cells: free energy bond J_{ij} where i and j are the types of the cells
- *dynamics:* Free energy minimization with volume, membrane conservation:

$$H = \sum \frac{J_{ij}}{2} + \sum J_{im} + \lambda(v - V)^2 + \alpha((m - M)^2$$

where $J_{i,j}$ represent surface energy between cells (and J_i, m that between a cell and the medium).
- Copy state of neighboring cell with probability:

$$P_{i->j} = 1 \qquad iff \quad \Delta H < -Y$$

$$P_{i->j} = e^{-(\Delta H + Y)/T} \qquad iff \quad \Delta H \geq -Y$$

where Y represents the energy expenditure of deformation (Yeild).

This energy based model automatically integrates multiple local forces on the cell. By measuring the deformation of cells these forces can be analyzed.

4 From cells to tissue level dynamics

Such a representation of a cell can generate a rich variety of tissue level dynamical properties which include:

- cell sorting by differential adhesion
- Individual cells 'wiggle' through cell mass
- Individual cells can 'move against the flow'
  e.g. by being smaller/larger; being in the minority or adhesion Käfer, Hogeweg & Marée 2006

5 Interfacing generic and informatic properties of cells

Important as the generic properties of biological cells are, they do not fully describe a biological cell. Cells are pre-ementenly information rich dynamical entities, which change there properties continuously, based on intra cellular and extra cellular signals. Thus to fully describe a cell, one needs to interface the mechanical properties with the intra cellular gene regulation dynamics.

Such an interface is easily achieved in the CPM formalism, because, although the basic dynamics is on the subcellular level, the cell exist as an entity in the model definition. The state of the dynamics of the intracellular dynamics can be mapped to the properties of the cell, e.g. its target volume (V), its cell surface properties (affecting $J_{i,j}$s) etc. By affecting the target volume (V) rather than the actual volume (v) the cell based dynamics co-determines if and how this state change will indeed change the cell volume. Likewise by changing surface receptors, the effect thereof will depend on the expressed receptors of neighboring cells.

Below is a list of potential ways in which the generic properties of the cells as defined above can be modified my informatic processes within the cell, together with some references to studies which have used this type of interfacing,

- Cell differentiation
  e.g. governed by gene regulation networks — *Hogeweg 2000a,b*
  leading to — ($J_{ij} -> J'_{ij}$)
  . — production of morphogens
  . — changes in Y,V,M

- Induced growth — (V++)
- cell division — $\sigma_i -> \sigma_i + \sigma_j$
- Squeeze induced Apoptosis — *Chen et al 1997* (λ small)
- Stretch induced growth — *Chen et al 1997* (if $v > V + \tau : V++$)
- intra and inter cellular reaction/diffusion systems . — *Savill and Hogeweg 1997*
  . — *Marée & Hogeweg 2001*
- Chemotaxis — ($\Delta H = \Delta H + grad$)
- Polarity: Persistent, adjustable directional motion — *Beltman et al 2007*
- explicit intracellular cytoskelon dynamics — *Marée et al 2006*
- particle based modeling of intracellular reaction dynamics — *Hogeweg & Takeuchi 2003*

In this way a great variety of developmental processes unfolding at many space and timescales can be studied by relatively simple models. For example the entire life cycle the slime mold *Dictyostelium discoideum*, which includes a single cell stage, a crawling multicellular slug which orients itself toward light and temperature, and finally settles down and metamorphoses to a fruiting body has been modeled in this way (Savill and

Hogeweg 1997, Marée and Hogeweg 1999, 2001). Another interesting example is the elucidation of the mechanism of the growth of plant roots, in particular *Arabidopsis*. It was shown (Grieneisen et al 2007) that the localization of surface receptors (PIN's) which pump auxin out of the cell, explains the formation of an auxin maximum near the tip of the root in a matter of seconds. Through this maximum and the resulting auxin gradients the cell division and elongation is regulated leading to the characteristic root growth and morphological changes over a time period of weeks. The shape of the cells turns out to be very essential for understanding the concentrations of auxin, and therewith the regulation of growth in the various regions of the root (Grieneisen et al in prep). Notwithstanding the relative simplicity of the model, the predictions derived from the in silico root have been successfully tested in the real plant

6 From cells to (evolved) mechanisms of morphogenesis

Not only do we want to understand the developmental mechanisms of particular extant organisms, but we also want to gain insight in generic mechanisms of morphogenesis as a result of the interface between mechanical properties of cells and the gene regulation. To this end yet an other timescale can be added to the model, i.e. the evolutionary timescale. Hogeweg (2000a,b) studied the morphogenetic mechanisms which emerged in a population of "critters" which developed from a single zygote to a multicolor clump of cells. The cells contained an initially random gene regulation network, which changed its structure over time by mutations, The selection criterion used was *cell differentiation*. By using this criterion, rather than morphogenesis itself as for the selection, "generic" morphogenetic mechanisms could be uncovered, i.e. those which emerged as side effect of cell differentiation.

Examples of the development of so evolved "critteres" are shown in fig 3. Thus, although the basic CPM formalism is an energy minimization formalism, which therefore tends to lead to "blobs" of cells, the interface with the dynamics of cell differentiation can lead to interesting morphologies. This is because the cell different ion process keeps the system out of equilibrium. The, by this model, uncovered mechanisms of morphogenesis resemble those described in multicolor organisms, both plants and animals. One interesting mechanism is "convergent extension" in which the polarization and elongation of cells in one direction, leads to tissue growth in the direction perpendicular to it (see fig 4).

7 Conclusions

Previous models of morphogenesis have often confined themselves to pattern formation (e.g. Turing patters) (e.g. Meinhardt anf Gierer 2000), or to purely informatic models of cells whose interactions are solely based on ancestry rather than on a dynamic neighbourhood of other cells (e.g. L systems) (Lindenmayer 1968, Hogeweg & Hesper 1974,

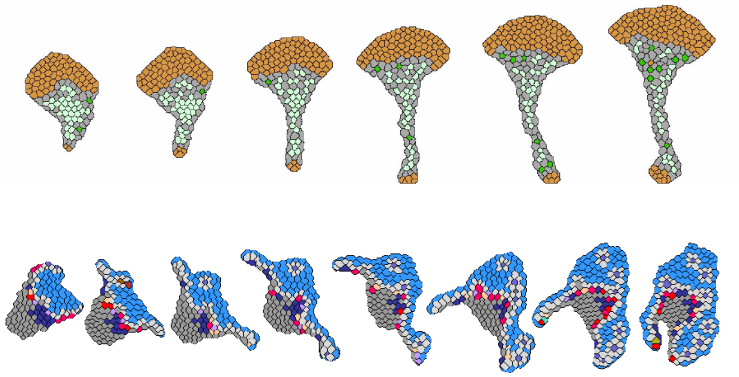


Fig. 6.3 evolution for cell differentiation leads to morphogenesis (Hogeweg 2000a,b)

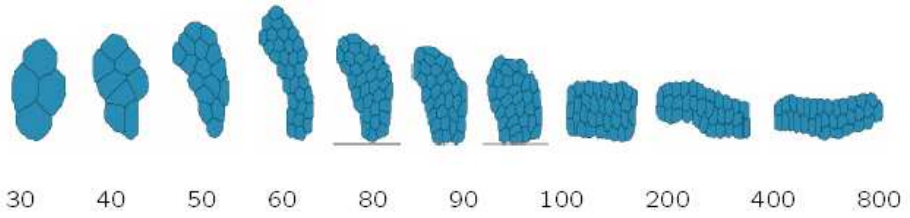


Fig. 6.4 convergent extension in bipolar cells (cf Zajac et al 2003, Hogeweg unpublished)

P. Prusinkiewicz and A. Lindenmayer 1990) The modeling formalism described here goes an important step beyond those approaches by combining generic (mechanical) properties of cells with informatic properties. This allows us to study morphogenesis in the strict sense: the generation of macroscopic shapes from subcellular processes. In the words of Segel(2001), referring to the *Dictyostelium* model, it allows us "to compute an organism".

We find in the variety of models studied that mesoscopic description of cell is essential, and that morphogenesis is a sustained out of equilibrium process through the interaction of processes at multiple space and time scales.

8 References

Beltman JB, Mare AF, Lynch JN, Miller MJ, de Boer RJ. Lymph node topology dictates T cell migration behavior. J Exp Med. 2007 Apr 16;204(4):771-80.

C.S. Chen, M. Mrksich, S. Huang, G.M. Whitesides and D.E. Ingber , Geometric control of cell life and death. Science 276 (1997), pp. 1425 1428.

Graner F, Glazier JA. Simulation of biological cell sorting using a two-dimensional extended Potts model. Phys Rev Lett. 1992 Sep 28;69(13):2013-2016

J.A. Glazier and F. Graner , Simulation of the differential driven rearrangement of biological cells. Phys. Rev. E 47 (1993), pp. 2128 21254.

Grieneisen VA, Xu J, Mare AF, Hogeweg P, Scheres B. Auxin transport is sufficient to generate a maximum and gradient guiding root growth. Nature. 2007 Oct 25;449(7165): 1008-13.

. Hogeweg and B. Hesper , A model study on morphological description. Pattern Recogn. 6 (1974)

Hogeweg P. Evolving mechanisms of morphogenesis: on the interplay between differential adhesion and cell differentiation. J Theor Biol. 2000 Apr 21;203(4):317-33.

Hogeweg P. Shapes in the shadow: evolutionary dynamics of morphogenesis. Artif Life. 2000 Winter;6(1):85-101.

Hogeweg P. Computing an organism: on the interface between informatic and dynamic processes. Biosystems. 2002 Jan;64(1-3):97-109.

Hogeweg P, Takeuchi N. Multilevel selection in models of prebiotic evolution: compartments and spatial self-organization. Orig Life Evol Biosph. 2003 Oct;33(4-5):375-403.

A. F. M. Maree, A. Jilkine, A. Dawes, V. A. Grieneisen, and L. Edelstein-Keshet. Polarization and movement of keratocytes: a multiscale modelling approach. Bull. Math. Biol., 68(5):1169 1211, July 2006

Käfer J, Hogeweg P, Marée AF. Moving forward moving backward: directional sorting of chemotactic cells due to size and adhesion differences. PLoS Comput Biol. 2006 Jun 9;2(6):e56. Epub 2006 Jun 9.

Lecuit T, Lenne PF. Cell surface mechanics and the control of cell shape, tissue patterns and morphogenesis. Nat Rev Mol Cell Biol. 2007 Aug;8(8):633-44. Review.

Lindenmayer A , Mathematical models for cellular interactions in development. I. Filaments with one-sided inputs. J. Theor. Biol. 18 3 (1968a), pp. 280 299.

Mar e AF, Panfilov AV, Hogeweg P. Migration and thermotaxis of dictyostelium discoideum slugs, a model study J Theor Biol. 1999 Aug 7;199(3):297-309.

Mare AF, Hogeweg P. How amoeboids self-organize into a fruiting body: multicellular coordination in Dictyostelium discoideum. Proc Natl Acad Sci U S A. 2001 Mar 27;98(7):3879-83.

A. F. M. Maree, A. Jilkine, A. Dawes, V. A. Grieneisen, and L. Edelstein-Keshet. Polarization and movement of keratocytes: a multiscale modelling approach. Bull. Math. Biol., 68(5):1169 1211, July 2006

Meinhardt and Gierer, 2000H. Meinhardt and A. Gierer , Pattern formation by local self-activation and lateral inhibition. Bioessays 22 8 (2000), pp. 753 760

Mombach JC, Glazier JA, Raphael RC, Zajac M. Quantitative comparison between differential adhesion models and cell sorting in the presence and absence of fluctuations. Phys Rev Lett. 1995 Sep 11;75(11):2244-2247.

P. Prusinkiewicz and A. Lindenmayer The Algorithmic Beauty of Plants, Springer, New York (1990).

Savill, N.J. and P. Hogeweg Modelling morphogenesis: from single cells to crawling slugs. J. theor. Biol. 1997 184, 229-235.

Segel L, Computing an organism. Proc. Natl. Acad. Sci. USA 98 7 (2001), pp. 3639 3640.

Zajac M, Jones GL, Glazier JA. Simulating convergent extension by way of anisotropic differential adhesion. J Theor Biol. 2003 May 21;222(2):247-59

Multi-Scale Modeling and Simulation of Cellular Membrane Transport and Reaction Systems

Markus Kirkilionis

University of Warwick, Mathematics Department,
Coventry CV4 7AL, UK
mak@maths.warwick.ac.uk

Based on the development of new experimental techniques, mainly imaging techniques like FRAP and FRET, cellular transport and reaction processes can be better understood than ever before. Cell biology in general offers a whole range of such important applications (in the end all life depends on this complex machinery) which are challenging for mathematical modeling and the performance of existing numerical algorithms. Inside the cell membrane systems like the ones involved in the secretory pathway are especially important and can be used to understand the state-of-the-art of modeling and simulation in a wider area to much advantage.

One specific property of cellular structures are their complicated geometry, here just called 'Complex Domains'. The complexity of these cellular domains can now be measured much more accurately with the help of modern imaging and image analysis techniques. It is tempting to combine methods from image analysis and numerical simulations in order to get a better understanding how different molecules, small to large (ions to protein complexes) distribute and react inside the cell. Moreover such geometries are typically 'complex' on every relevant scale. This makes their representation in a simulation typically difficult, and one must use adequate approximations. This need for averaging nicely explains why (in this case spatial) scales are important (and why all modeling is relative to a chosen scale), and therefore scale is perhaps the most important notion in this area, even before considering the appropriate choice of either a discrete or continuous state space to represent the different components of the system.

The same is true for the second part of the problem, the action (mostly binding) of macro-molecules with each other or other binding partners. This process creates events at which the system state changes, and that again needs to be represented relative to a chosen temporal scale in the model (and finally the simulation). We discuss this with the help of birth-death processes and the dynamics of Markov chains, and describe the dynamics of ion channels in a typical membrane with the help of multi-scale analysis.

A Multiscale Modelling Framework Based On P Systems

Francisco J. Romero-Campero

University of Nottingham, School of Computer Sciences and IT,
Jubilee Campus, Nottingham, NG8 1BB, UK
fxc@cs.nott.ac.uk

Cellular systems present a highly complex organisation at different scales including the molecular, cellular and colony levels. The complexity at each one of these levels is tighly interrelated to each other. Integrative systems biology aims to obtain a deeper understanding of cellular systems by focusing on the systemic integration of the different levels of organisation in cellular systems.
The different approaches in celluar modelling within systems biology have been classified into mathematical and computational frameworks. Specifically, the methodology to develop computational models has been recently called executable biology since it produces executable algorithms whose computations resemble the evolution of cellular systems.
In this work we present P systems as a multiscale modelling framework within executable biology. P system models explicitly specify the molecular, cellular and colony levels in cellular systems in a relevant and understandable manner. Molecular species and their structure are represented by objects or strings, compartmentalisation is described using membrane structures and finally cellular colonies and tissues are modelled as a collection of interacting individual P systems. The interactions between the components of cellular systems are described using rewriting rules. These rules can in turn be grouped together into modules to characterise specific cellular processes.
One of our current research lines focuses on the design of cell systems biology models exhibiting a prefixed behaviour by assembling automatically these cellular modules. Our approach is equally applicable to systems as well as synthetic biology.

Interactive processes in systems based on facilitation and inhibition

Grzegorz Rozenberg

Universiteit Leiden, LIACS,
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
rozenber@liacs.nl

From a behavioural point of view many systems encountered in nature are based on interactions between a number (usually a huge number) of basic processes. Often such interactions are based on two main mechanisms: facilitation and inhibition. For example, this point of view is well suited for considering systems of biochemical reactions, where individual reactions influence each other through facilitation/acceleration and inhibition/retardation.
In our talk we discuss reaction systems which constitute a formal framework for specifying, modeling, and analyzing the kind of systems outlined above.
In particular we will discuss the basic axioms (assumptions) that hold for a great number of biochemical reactions, and therefore underlie the formal setup for reaction systems. We point out that these axioms are quite different (often orthogonal) to the underlying axioms of traditional/typical models considered in theoretical computer science.
We will present a number of formal properties of reaction systems as well as a number of research topics/areas, such as modularity and introducing time in reaction systems. The talk is self-contained; in particular we do not require any specific knowledge of basic properties of biochemical reactions.

On the Qualitative Analysis of Conformon P-Systems

Parosh Aziz Abdulla[1], Giorgio Delzanno[2], Laurent Van Begin[3]

[1]Uppsala University, Department of Information Technology,
Lägerhyddsvägen 2, 752 37 Uppsala, Sweden
parosh@it.uu.se

[2]Università di Genova, Dipartimento di Informatica e Scienze dell'Informazione,
via Dodecaneso 35, 16146 Genova, Italy
giorgio@disi.unige.it

[3]Université Libre de Bruxelles, Département d'Informatique,
Boulevard du Triomphe, 1050 Bruxelles, Belgium
lvbegin@ulb.ac.be

We study computational properties of conformon P-systems, an extension of P-systems in which symbol objects are labelled by their current amount of energy. We focus here our attention to decision problems like reachability and coverability of a configuration and give positive and negative results for the full model and for some of its fragments. Furthermore, we investigate the relation between conformon P-systems and other concurrency models like *nested Petri nets* and *constrained multiset rewriting systems*.

1 Introduction

P-systems [10] are a basic model of the living cell defined by a set of hierarchically organized membranes and by rules that dynamically distribute elementary objects in the component membranes. Conformon P-Systems [5] are an extension of P-systems in which symbol objects (conformons) are labelled with their current amount of energy. In a conformon P-system membranes are organized into a directed graph. Furthermore, a symbol object is a pair name-value, where name ranges over a given alphabet, and value is a natural number. The value associated to a conformon denotes its current amount of energy. Conformon P-systems provide rules for the exchange of energy from a conformon to another and for passing through membranes. Passage rules are conditioned by predicates defined over the values of conformons. In [6] Frisco and Corne applied conformon P-systems to model the dynamics of HIV infection. Concerning the expressive power of conformon P-systems, in [5] Frisco has shown that the model is Turing equivalent even without the use of priority or maximal parallelism.

In this paper we investigate restricted fragments of conformon P-systems for which decision problems related to verification of qualitative properties are decidable. We focus our attention to verification of safety properties and decision problems like coverability of a configuration [1]. The fragment we consider put some restrictions on the form of predicates used as conditions of passage rules. Namely, we only admit passage rules with lower bound constraints on the amount of energy as conditions (i.e. $p(x) \stackrel{\text{def}}{=} x \geq c$ for $c \in \mathbb{N}$). The resulting fragment, we will refer to as *restricted conformon P-systems*, is still interesting as a model of natural processes. Indeed, we can use it to specify systems in which conformons pass through a membrane when a given amount of energy is reached.

For restricted conformon P-systems, we apply the methodology of [1] to define an algorithm to decide the coverability problem. This algorithm performs a backward reachability analysis through the state space of a system. Since in our model the set of configurations is infinite, the analysis is made symbolic in order to finitely represent infinite sets of configurations. For this purpose, we use the theory of *well-quasi orderings* and its application to verification of concurrent systems [1].

In the paper we also investigate the relation between (restricted) conformon P-systems and other models used in concurrency like Petri nets [11], nested Petri nets [8], and constrained multiset rewriting systems (CMRS) [2]. Specifically, we show that conformon P-systems are a special class of nested Petri nets, and restricted P-systems are a special class of CMRS. This comparison gives us indirect proofs for decidability of coverability in restricted conformon P-systems that follows from the results obtained for nested nets and CMRS in [9, 2].

To our knowledge, this is the first work devoted to the analysis of problems like coverability for conformon P-systems, and to the comparison of the same models with other concurrency models like nested Petri nets and CMRS.

Plan of the paper In Section 2 we introduce the conformon P-systems model. In Section 3 we study decision problems like reachability and coverability. In Section 4 we compare conformon P-systems with nested Petri nets and CMRS. Finally, in Section 5 we discuss related work and address some conclusions.

2 Conformon P-systems

Let V be a finite alphabet and $\mathbb{N}$ the set of natural numbers. A *conformon* is an element of $V \times \mathbb{N}_0$ where $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, denoted by $[X, x]$. We will refer to X as the *name* of the conformon $[X, x]$ and to x as its *value*. In the rest of the paper we work with multisets of conformons. We use $\{\!\{a_1, \ldots, a_n\}\!\}$ to indicate a multiset with elements $a_1, \ldots, a_n$, and symbols $\oplus$ and $\ominus$ to indicate resp. multiset union and difference. We use $\mathcal{C}_V$ to denote the set of conformons defined over alphabet V.

Conformons are situated inside a finite set of membranes or regions. Let N be the set of membrane names. A *configuration* μ is a tuple (indexed on N) of multisets of conformons. For simplicity we often assume that membranes are numbered from 1 to n and that configurations are tupled $(\xi_1, \ldots, \xi_n)$ where ξ_i is a multiset of conformons in $\mathcal{C}_V$.

The dynamic behavior of conformons is described via a set of rules of the following form:

- A *creation* rule has the form $\stackrel{e}{\searrow}_m A$, where $A \in V$, $e \in \mathbb{N}_0$, and $m \in N$ and defines the creation of a conformon $[A, e]$ inside membrane m. A creation rule for conformon $[A, e]$ in membrane m corresponds to a conformon $[A, e]$ with cardinality ω in [5]. The use of creation rules allows us to obtain a better comparison with other Petri net models as discussed later in the paper.
- An *internal* rule has the form $A \stackrel{e}{\rightarrow}_m B$, where $A, B \in V$, $e \in \mathbb{N}$, $m \in N$ and defines the passage of a quantity e of energy from a conformon of type A to one of type B inside membrane m.
- A *passage* rule has the form $m \stackrel{p}{\hookrightarrow} n$ where $m, n \in N$ and $p(x)$ is a monadic predicate of one of the following forms $x = a$, $x \geq a$, $x \leq b$ for $a \in \mathbb{N}_0$ and $b \in \mathbb{N}$. With this rule, a conformon $[X, x]$ inside m can move to membrane n if $p(x)$ is satisfied by the current value of X.

As in tissue P-systems, the underlying structure of membranes is here a finite graph whose nodes are the membranes in N and edges are defined by passage rules. We are ready now for a formal definition of conformon P-systems.

Definition 1. (Conformon P-system) *A* basic conformon P-system of degree $m \geq 1$ with unbounded values *(*c*P-system for short) is a tuple* $\Pi = (V, N, R, \mu_0)$*, V is a finite set of conformon names, N is a finite set of membranes names (we assume that each membrane has a distinct name), R is a set of rules, μ_0 is an initial configuration.*

Given a configuration μ, we say that an internal rule $r = A \stackrel{e}{\rightarrow}_m B$ is enabled at μ if there exist a conformon $[A, x] \in \mu(m)$ and a conformon $[B, y] \in \mu(m)$ such that $x \geq e$; we say in this case that r operates on conformons $[A, x]$ and $[B, y]$ in μ. A passage rule $r = m \stackrel{p}{\hookrightarrow} n$ is enabled at μ if there exists a conformon $[A, x] \in \mu(m)$ such that $p(x)$ is satisfied; we say here that r operates on conformon $[A, x]$ in μ. Notice that creation rules are always enabled. The evolution of a conformon P-system Π is defined via a transition relation $\Rightarrow$ defined on configurations as follows. A configuration μ may evolve to μ', written $\mu \Rightarrow \mu'$, if one of the following conditions is satisfied:

- There exists a rule $r = A \stackrel{e}{\rightarrow}_m B$ in R which is enabled in μ and operates on conformons $[A, x]$ and $[B, y]$, and the following conditions are satisfied:
  - $\mu'(m) = (\mu(m) \ominus \{\!\{[A, x], [B, y]\}\!\}) \oplus \{\!\{[A, x - e], [B, y + e]\}\!\}$;

  - $\mu'(n) = \mu(n)$ for any $n \neq m$.
- There exists a rule $r = m \overset{p}{\hookrightarrow} n$ in R which is enabled in μ and operates on conformon $[A, x]$ (i.e. $p(x)$ is true) and the following conditions are satisfied:
  - $\mu'(m) = \mu(m) \ominus \{\!\{[A, x]\}\!\}$;
  - $\mu'(n) = \mu(n) \oplus \{\!\{[A, x]\}\!\}$;
  - $\mu'(p) = \mu(p)$ for any $p \neq m, n$.
- There exists a rule $r = \overset{e}{\underset{m}{\Rightarrow}} A$ in R and the following conditions are satisfied:
  - $\mu'(m) = \mu(m) \oplus \{\!\{[A, e]\}\!\}$;
  - $\mu'(p) = \mu(p)$ for any $p \neq m$.

In the rest of the paper we use $\Rightarrow^*$ to indicate the reflexive and transitive closure of the transition relation $\Rightarrow$. Furthermore, we say that μ evolves into μ' if $\mu \Rightarrow^* \mu'$, i.e., there exists a finite sequence of configurations $\mu_1, \ldots, \mu_r$ such that $\mu = \mu_1 \Rightarrow \ldots \Rightarrow \mu_r = \mu'$. Furthermore, given a set of configurations S, the set of successor configurations is defined as

$$Post(S) \overset{\text{def}}{=} \{\mu' \mid \mu \Rightarrow \mu',\ \mu \in S\}$$

and the set of predecessor configurations is defined as

$$Pre(S) \overset{\text{def}}{=} \{\mu' \mid \mu' \Rightarrow \mu,\ \mu \in S\}$$

Notice that the transition relation $\Rightarrow$ defines an interleaving semantics for a *c*P-system Π, i.e., only a single rule among those enabled can be fired at each evolution step of Π. This semantics is slightly different from the original semantics in [5] where an arbitrary subset of all enable rules can be fired at each evolution step. It is important to remark however that the two semantics are equivalent with respect to the kind of qualitative properties (reachability problems) we consider in this paper.

As an example, consider the *c*P-system with two membranes m_1 and m_2 and $N = \{A, B, C\}$, and with the rules $\overset{1}{\underset{m_1}{\Rightarrow}} A$, $A \overset{1}{\underset{m_1}{\Rightarrow}} B$, and $m_1 \overset{p}{\hookrightarrow} m_2$ where $p(x)$ is defined by the equality $x = 3$. In this model the configuration $c = (\{\!\{[B, 0]\}\!\}, \emptyset)$ may evolve as follows:

$$\begin{array}{l} c \Rightarrow (\{\!\{[A,1],[B,0]\}\!\},\emptyset) \Rightarrow (\{\!\{[A,1],[A,1],[B,0]\}\!\},\emptyset) \Rightarrow \\ (\{\!\{[A,1],[A,1],[A,1],[B,0]\}\!\},\emptyset) \Rightarrow (\{\!\{[A,1],[A,1],[A,0],[B,1]\}\!\},\emptyset) \Rightarrow \\ (\{\!\{[A,1],[A,0],[A,0],[B,2]\}\!\},\emptyset) \Rightarrow (\{\!\{[A,0],[A,0],[A,0],[B,3]\}\!\},\emptyset) \Rightarrow \\ (\{\!\{[A,0],[A,0],[A,0]\}\!\},\{\!\{[B,3]\}\!\}) \end{array}$$

Finally, notice that both our semantics and Frisco's semantics in [5] do not require all enabled rules to be fired simultaneously as in the semantics of P-systems (maximal parallelism). In general, maximal parallelism and interleaving semantics may lead to models with different computational power.

3 Qualitative analysis of *c*P-systems

In [5] Frisco introduced the class of *c*P-systems with *bounded values* in which the only type of admitted creation rules have the form $\frac{0}{m}A$, i.e., the only type of conformons for which there is no upper bound on the number of occurrences in reachable configurations (finite but unbounded multiplicity) are of the form $[A, 0]$. In *c*P-system with *bounded values* the total amount of energy in the system is always constant. Thus, with this restriction, the only dimension of infiniteness of the state-space is the number of occurrences of conformons. This kind of restricted systems, say *c*P-systems with bounded values, can be represented as Petri nets. Thus, several interesting qualitative properties like reachability and coverability of a configuration can be decided for this fragment of *c*P-systems.

In the full model the set of configurations reachable from an initial one may be infinite in two dimensions, i.e., in the number of conformons occurring in the membrane system and in the amount of total energy exchanged in the system. In [5] Frisco has proved that full *c*P-systems are a Turing equivalent model. Despite of the power of the model, we prove next that a basic qualitative property called *reachability* can be decided for full *c*P-systems. Let us first define the reachability problem.

Definition 2. (Reachability problem)
The reachability problem is defined as follows: Given a c*P-system* $\Pi = (V, N, R, \mu_0)$ *and a configuration* μ_1*, does* $\mu_0 \Rightarrow^* \mu_1$ *hold?*

The following results then hold.

Theorem 1. (Decidability of reachability for full *c*P-systems)
The reachability problem (w.r.t. relation $\Rightarrow$*) is decidable for any* c*P-system.*

Proof The proof is based on a reduction of reachability of configuration μ_1 in a *c*P-system Π to reachability in a finite-state system extracted from Π and μ_1. The reduction is based on the following key observation. For two configurations μ_0 and μ_1 the set Q of distinct configurations that may occur in all possible evolutions from μ_0 to μ_1 is finite. This property is a consequence of the fact that internal and passage rules maintain constant the total number of conformons and the total amount of energy of a system (sum of the values of all conformons) whereas creation rules may only increase both parameters. Thus, the total amount of conformons and of energy in configuration μ_1 gives us an upper bound U_C on the possible number of conformons and an upper bound U_V on their corresponding values in any evolution from μ_0 to μ_1. Based on this observation, it is simple to define a finite-state automaton S with states in Q and transition relation δ defined by instantiating the rules in R over the elements in Q. As an example, if $V = \{A, B\}$, $N = \{m, n\}$, $U_C = 10$ and $U_V = 4$ and R contains

the rule $r = A\frac{2}{m}B$. Then, we have to consider a finite state automaton in which the states are all possible configurations containing at most 10 elements taken from the alphabet $\Sigma = \{[X, n] \mid X \in V, 0 \leq n \leq 4\}$. The rule r generates a transition relation δ that put in relations two states μ and μ' iff $\mu(m)$ contains a pair of elements $[A, a], [B, b] \in \Sigma$ such that a and $a + 2$ satisfy the condition $2 \leq a, a + 2 \leq 4$, $\mu'(m) = (\mu(m) \ominus \{\!\{[A, a], [B, b]\}\!\}) \oplus \{\!\{[A, a - 2], [B, b + 2]\}\!\}$ and $\mu'(m') = \mu(m')$ for all the membranes $m' \neq m$. The finite automaton S satisfies the property that μ_1 is reachable from μ_0 in the *c*P-system Π if and only if the pair (μ_0, μ_1) is in the transitive closure of δ. The thesis then follows from the decidability of configuration reachability in a finite-automaton. □

In order to study verification of safety properties, we need to introduce an ordering between configurations similar to the coverability ordering used for models like Petri nets. We use here an ordering $\sqsubseteq$ between configurations μ and μ' such that for each membrane m, each conformon in $\mu(m)$ is mapped to a distinguished conformon in $\mu'(m)$ that has the same name and greater or equal value. This ordering allows us to reason about the presence of a conformon with a given name and at least a given amount of energy inside a configuration.

Example 1 Consider the configurations

$$\mu_1 = (\{\!\{[A, 2], [A, 4], [B, 3]\}\!\}, \{\!\{[A, 5]\}\!\})$$
$$\mu_2 = (\{\!\{[A, 4], [A, 5], [B, 6], [C, 8]\}\!\}, \{\!\{[A, 7], [B, 5]\}\!\})$$

Then $\mu_1 \sqsubseteq \mu_2$, since $[A, 2]$, $[A, 4]$ and $[B, 3]$ in membrane 1 of μ can be associated resp. to the conformons $[A, 4]$, $[A, 5]$ and $[B, 6]$ in membrane 1 of μ_2; furthermore, $[A, 5]$ in membrane 2 of μ can be associated to conformon $[A, 7]$ in membrane 2 of μ_2. Consider now the configurations

$$\mu_3 = (\{\!\{[A, 4], [A, 5], [B, 1]\}\!\}, \{\!\{[A, 7], [B, 5]\}\!\})$$
$$\mu_4 = (\{\!\{[A, 5], [B, 6]\}\!\}, \{\!\{[A, 7], [B, 5]\}\!\})$$

Then, $\mu_1 \not\sqsubseteq \mu_4$ since there is no conformon in membrane 1 in μ_3 with name B and value greater or equal than 3. Furthermore, $\mu_1 \not\sqsubseteq \mu_4$ since we cannot associate two different conformons, namely $[A, 2]$ and $[A, 4]$ in μ_1, to the same conformon, namely $[A, 5]$, in μ_4. Finally, notice that the configuration $\mu = (\{\!\{[A, 0]\}\!\}, \emptyset)$ is such that $\mu \sqsubseteq \mu_i$ for $i : 1, \ldots, 4$. The configuration μ can be used to characterize the presence of a conformon with name A in membrane 1 no matter of how energy it has.

The ordering $\sqsubseteq$ is formally defined as follows.

Definition 3. (Ordering $\sqsubseteq$) *Given two configurations μ and μ', $\mu \sqsubseteq \mu'$ iff for each $m \in N$ there exists an* injective mapping h_m *from $\mu(m)$ to $\mu'(m)$ that satisfies the following condition: for each $[A, x] \in \mu(m)$, if $h_m([A, x]) = [B, y]$, then $A = B$ and*

$x \leq y$ *(*$[A, x]$ *is associated to a conformon with the same name and larger amount of energy).*

A set S of configurations is said *upward closed* w.r.t. $\sqsubseteq$ if the following condition is satisfied: for any $\mu \in S$, if $\mu \sqsubseteq \mu'$ then $\mu' \in S$. In other words if a configuration μ belongs to an upward closed set S then all the configurations greater than μ w.r.t. $\sqsubseteq$ belong to S either.

Consider now the following decision problem.

Definition 4. (Coverability problem) *The coverability problem is defined as follows: Given a* c*P-system* $\Pi = (V, N, R, \mu_0)$ *and a configuration* μ_1*, is there a configuration* μ_2 *such that* $\mu_0 \Rightarrow^* \mu_2$ *and* $\mu_1 \sqsubseteq \mu_2$*?*

Coverability can be viewed as a weak form of *configuration reachability* in which we check whether configurations with certain constraints can be reachable from the initial configuration. In concurrency theory, the coverability problem is strictly related to the verification of safety properties. This link can naturally be transferred to *qualitative properties* of natural systems. As an example, checking if a configuration in which two conformons with name A can occur in membrane m during the evolution of a system amounts to checking the coverability problem for the target configuration μ_2 defined as $\mu_2(m) = \{\!\{[A, 0], [A, 0]\}\!\}$ and $\mu_2(m') = \emptyset$ for $m' \neq m$. The following negative result then holds.

Proposition 1 *The coverability problem is undecidable for full* c*P-systems.*

Proof The encoding of a counter machine M in *c*P-systems can be adapted to our formulation with creation rules in a direct way: conformons with ω-cardinality are specified here by creation rules. In the encoding in [5] an execution of the counter machine M leading to location ℓ is simulated by the evolution of a *c*P-system Π_M that reaches a configuration containing a conformon $[\ell, 9]$ in a particular membrane, say m. Notice also that the membrane m cannot contain, by construction, a conformon $[\ell, v]$ with $v > 9$. Thus, coverability of the configuration with $[\ell, 9]$ inside m in Π_M corresponds to reachability of location ℓ in M. Since location reachability is undecidable for counter machines, coverability is undecidable for *c*P-systems. □

3.1 A syntactic fragments of *c*P-systems In this section we show that checking safety properties can be decided for a fragment of *c*P-systems with a restricted form of passage rules in which conditions are only defined by lower bound constraints.

Definition 5. (Restricted *c*P-systems) *We call* restricted *the fragment of* c*P-systems in which we forbid the use of predicates of the form $x = c$ and $x \leq c$ as conditions of passage rules.*

Our main result is that, despite of the two dimension of infiniteness, the coverability problem is decidable for restricted *c*P-systems with an arbitrary number of conformons. To prove the result we adopt the methodology proposed in [1], i.e., we first show that restricted *c*P-systems are monotonic w.r.t. $\sqsubseteq$. We then show that $\sqsubseteq$ is a well-quasi ordering. This implies that any upward closed set is represented via a finite set of minimal (w.r.t. $\sqsubseteq$) configurations. Thus, minimal elements can be used to finitely represent infinite (upward closed) sets of configurations. Finally, we prove that, given an upward closed set S of configurations, it is possible to compute a finite representation of the set of predecessor configurations of S. Monotonicity ensures us that such a set is still upward closed. We compute it by operating on the minimal elements of S only.

Lemma 1. (Monotonicity) *Restricted* c*P-systems are monotonic w.r.t. $\sqsubseteq$, i.e., if $\mu_1 \Rightarrow \mu_2$ and $\mu_1 \sqsubseteq \mu_1'$, then there exists μ_2' such that $\mu_1' \Rightarrow \mu_2'$ and $\mu_2 \sqsubseteq \mu_2'$.*

Proof Let μ_1 be a configuration evolving into μ_2, and let $\mu_1 \leq \mu_1'$. The proof is by case analysis on the type of rules applied in the execution step. Notice that the case of creation rule is trivial. Hence, we concentrate on the two remaining cases.
Internal rule. Let us consider a single application of an internal rule $A \xrightarrow[m]{e} B$ operating on conformons $[A, x]$ and $[B, y]$ in membrane m. Since the rule is enabled we have that $x \geq e$. Furthermore, the application of the rule modifies the value of the two conformons as follows: $[A, x - e]$ and $[B, y + e]$.
Since $\mu_1 \sqsubseteq \mu_1'$ and by definition of $\sqsubseteq$, we have that there exist conformons $[A, x']$ and $[B, y']$ in membrane m of μ_1' such that $x \leq x'$ and $y \leq y'$. Thus, the same rule can be applied to $[A, x']$ and $[B, y']$ leading to a configuration μ_2' in which the two selected conformons are updated as follows: $[A, x' - e]$ and $[B, y' + e]$. Finally, we notice that, since $x' \geq x \geq e$, we have that $x - e \leq x' - e$ and $y + e \leq y' + e$. Thus, $\mu_2 \sqsubseteq \mu_2'$.
Passage rule. Let us consider a single application of a passage rule $m \xrightarrow{p} n$ with $p(y) \stackrel{\text{def}}{=} y \geq e$ and operating on the conformons $[A, x]$ in membrane m. Since the rule is enabled we have that $x \geq e$. Furthermore, the application of the rule moves the conformon to membrane n in μ_2.
Since $\mu_1 \sqsubseteq \mu_1'$ and by definition of $\sqsubseteq$, we have that there exist conformons $[A, x']$ in membrane m of μ_1' such that $x \leq x'$. Thus, the same passage rule is enabled in μ_1' and can be applied to move $[A, x']$ in membrane n in μ_2'.

Thus, we have that $\mu_2 \sqsubseteq \mu_2'$. □

From the monotonicity property, we obtain the following corollary.

Corollary 1 *For any restricted* c*P-systems and any upward closed set (w.r.t.* $\sqsubseteq$*)* S *of configurations, the set of predecessor configurations of* S*, namely* $Pre(S) \stackrel{\text{def}}{=} \{\mu \mid \mu \Rightarrow \mu',\ \mu' \in S\}$*, is upward closed.*

It is important to notice that the last two properties do not hold for full *c*P-systems. As an example, a passage rule from membrane 1 to 2 with predicate $x = 0$ is not monotonic w.r.t. to the configurations $\mu_1 = (\{\!\{[A,0]\}\!\}, \emptyset)$ and $\mu_1' = (\{\!\{[A,1]\}\!\}, \emptyset)$. Indeed, $\mu_1 \sqsubseteq \mu_1'$ and $\mu_1 \Rightarrow \mu_2 = (\emptyset, \{\!\{[A,0]\}\!\})$ but μ_1' has no successors. Furthermore, the set of predecessors of the upward closed set with minimal element μ_3 is the singleton containing μ_1 (clearly not an upward closed set).

Let us now go back to the properties of the ordering $\sqsubseteq$. We first have the following property.

Lemma 2 *Given a* c*P-system* Π *and two configurations* μ *and* μ'*, checking if* $\mu \sqsubseteq \mu'$ *holds (i.e. if* μ *is more general than* μ'*) is a decidable problem.*

Indeed, to decide it we have to select an appropriate injective mapping from a a finite set of mappings from μ to μ' and, then, to compute a finite set of multiset inclusions.

Let us now recall the notion of *well-quasi ordering* (see e.g. [7]).

Definition 6. ($\sqsubseteq$ **is a wqo)** *A quasi ordering* $\preceq$ *on a set* S *is a well-quasi ordering (wqo) if and only if for any infinite sequence* $a_1, a_2, \ldots$ *of elements in* S *(i.e.* $a_i \in S$ *for any* $i \geq 1$*) there exist indexes* $i < j$ *such that* $a_i \preceq a_j$*.*

The following important property then holds.

Lemma 3. ($\sqsubseteq$ **is a wqo)** *Given a* c*P-system* $\Pi = (V, N, R, \mu_0)$*, the ordering* $\sqsubseteq$ *defined on the set of all configuration of* Π *is a wqo.*

Proof Assume $N = \{1, \ldots, m\}$ as the set of membrane names. Let us first notice that a configuration μ can be viewed as a multiset of multisets of objects over the alphabet $V^1 \cup \ldots \cup V^m$, where $V^i = \{v^i \mid v \in V\}$. Indeed, μ can be reformulated as the multiset union $\rho_1 \oplus \ldots \oplus \rho_m$ where for each $[A, x] \in \mu(m)$, ρ_i contains a multiset with x occurrences of A^m. E.g., $\mu_1 = (\{\!\{[A,2],[B,3]\}\!\}, \{\!\{[A,5]\}\!\})$ can be reformulated as the multiset of multisets $\{\!\{\{\!\{A^1, A^1\}\!\}, \{\!\{B^1, B^1, B^1\}\!\}, \{\!\{A^2, A^2, A^2, A^2, A^2\}\!\}\}\!\}$.

When considering the aforementioned reformulation of configurations, the ordering $\sqsubseteq$ corresponds to the composition of multiset embedding (the existence of injective mapping $h_1, \ldots, h_m$) and multiset inclusion (the constraint on values). Since multiset in-

clusion is a well-quasi ordering, we can apply Higman's Lemma [7] to conclude that $\sqsubseteq$ is a well-quasi ordering. □

As a consequence of the latter property, we have that every upward closed set S of configurations is generated by a finite set of minimal elements, i.e., for any upward closed set S there exists a finite set F of configurations such that $S = \{\mu' \mid \mu \leq \mu', \mu \in F\}$. F is called the *finite basis* of S. As proved in the following lemma, given a finite basis of a set S, it is possible to effectively compute the finite basis of $Pre(S)$.

Lemma 4. (Computing Pre) *Given a finite basis F of a set S, there exists an algorithm that computes a finite basis F' of $Pre(S)$.*

Proof The algorithm is defined by cases as follows.

Creation rules Assume $\frac{e}{m}A \in R$ and $\mu \in F$. Then, μ occurs in F'. Furthermore, suppose that $\mu(m)$ contains a conformon $[A, e]$. Then, F' also contains the configurations μ' that satisfies the following conditions:

- $\mu'(m) = \mu(m) \ominus \{\!\{[A, e]\}\!\}$;
- $\mu'(n) = \mu(n)$ for $m \neq n$.

Internal rules Assume a rule $r = A\frac{e}{m}B \in R$ and $\mu \in F$. We have several cases to consider.

- We first have to consider a possible application of r to two conformons that are not explicitly mentioned in μ. This leads to a predecessor configuration in which we require at least the presence of A with at least value e and the presence of B with any value. Thus, F' contains the configurations μ' that satisfies the following conditions:
  - $\mu'(m) = \mu(m) \oplus \{\!\{[A, e], [B, 0]\}\!\}$;
  - $\mu'(n) = \mu(n)$ for $m \neq n$.
- We now have to consider the application of r to a conformon A with value x in μ and to a conformon B not explicitly mentioned in μ. This leads to a predecessor configuration in which we require at least the presence of A with at least value $x + e$ and the presence of B with any value. Thus, if $[A, x] \in \mu(m)$, F' contains the configurations μ' that satisfies the following conditions:
  - $\mu'(m) = (\mu(m) \ominus \{\!\{[A, x]\}\!\}) \oplus \{\!\{[A, x + e], [B, 0]\}\!\}$;
  - $\mu'(n) = \mu(n)$ for $m \neq n$.
- Furthermore, we have to consider the application of r to a conformon B with value $y \geq e$ in μ and to a conformon A not explicitly mentioned in μ. This leads to a predecessor configuration in which we require at least the presence of A with at

least value e and the presence of B with value $y - e$. Thus, if $[B, y] \in \mu(m)$ and $y \geq e$, F' contains the configurations μ' that satisfies the following conditions:
  * $\mu'(m) = (\mu(m) \ominus \{\!\{[B, y]\}\!\}) \oplus \{\!\{[A, e], [B, y - e]\}\!\}$;
  * $\mu'(n) = \mu(n)$ for $m \neq n$.
- Finally, we have to consider the application of r to a conformon B with value $y \geq e$ and to a conformon A with value x both in μ. This leads to a predecessor configuration in which we require at least the presence of A with at least value $x + e$ and the presence of B with value $y - e$. Thus, if $[A, x], [B, y] \in \mu(m)$ and $y \geq e$, F' contains the configurations μ' that satisfies the following conditions:
  * $\mu'(m) = (\mu(m) \ominus \{\!\{[A, x], [B, y]\}\!\}) \oplus \{\!\{[A, x + e], [B, y - e]\}\!\}$;
  * $\mu'(n) = \mu(n)$ for $m \neq n$.

Passage rules Assume $m \overset{p}{\hookrightarrow} n \in R$ with $p(x)$ defined by $x \geq e$ and $\mu \in F$. We first have to consider a possible application of r to a conformon that is not explicitly mentioned in μ. This leads to a predecessor configuration in which we require at least the presence of A with at least value e in membrane m. Thus, F' contains the configurations μ' that satisfies the following conditions:

- $\mu'(m) = \mu(m) \oplus \{\!\{[A, e]\}\!\}$;
- $\mu'(n) = \mu(n)$ for $m \neq n$.

Furthermore, suppose that $\mu(n)$ contains a conformon $[A, x]$ with $x \geq 0$. Then, F' contains the configurations μ' that satisfies the following conditions:

- $\mu'(n) = \mu(n) \ominus \{\!\{[A, x]\}\!\}$;
- $\mu'(m) = \mu(m) \oplus \{\!\{[A, v]\}\!\}$;
- $\mu'(p) = \mu(p)$ for $p \neq m, n$.

where $v = max(e, x)$ (the maximum between e and x).
The correctness follows from a case analysis. □

Theorem 2. (Decidability of Coverability for Restricted *c*P-systems) *The coverability problem is decidable for restricted* c*P-systems.*

Proof The thesis follows from Lemmas 1, 3, 4, and from [1, Theorem 4.1]. □

4 Relation with other models

In this section we compare *c*P-systems with other models used in the concurrency field, namely the nested Petri nets of [8] and the constrained multiset rewriting systems (CMRS) of [2].

4.2 *c*P-systems vs nested Petri nets Let us first recall that a Petri net (P/T system) [11] is a tuple (P, T, m_0) where P is a finite set of *places*, T is a finite set of *transitions*, and m_0 is the initial marking. Intuitively, places correspond to location or states of a given system. Places are populated with tokens, i.e., indistinguishable objects, that can be used e.g. to mark a given set of states to model concurrent processes. Tokens have no internal structure. This means that we are only interested in the multiplicity of tokens inside a place. Transitions are used to control the flow of tokens in the net (they define links between different places and regulate the movement of tokens along the links). More formally, a *transition* t has a pre-set ${}^\bullet t$ and a post-set $t^\bullet$ both defined by multisets of places in P. A marking is just a multiset with elements in P, a mapping from P to non-negative integers. Given a marking m and a place p, we say that the place p contains $m(p)$ *tokens*. A transition t is enabled at the marking m if ${}^\bullet t$ is contained as a sub-multiset in m. If it is the case, firing t produces a marking m', written $m \xrightarrow{t} m'$, defined as $(m \ominus {}^\bullet t) \oplus t^\bullet$.

A Petri net with inhibitor arc is a Petri net in which transitions can be guarded by an emptyness test on a subset of the places. For instance, a transition with an inhibitor arc on place p is enabled only when p is empty.

Nested Petri nets Differently from P/T systems, in a *nested Petri net* tokens have an internal structure that can be arbitrarily complex (e.g. a token can be a P/T system, or a P/T system with tokens that are in turn P/T systems, and so on). For instance, a *2-level nested Petri net* is defined by a P/T system that describes the whole system, called *system net*, and by a P/T system that describes the internal structure of tokens, called *element net*.

The transitions of the system net can be used to manipulate tokens as black boxes, i.e. without changing their internal structure. These kind of transitions are called *transport rules* (they move complex objects around the places of the system net).
Transitions of the element nets can be used to change the internal structure of a token without changing the marking of the system net. These kind of transitions are called *autonomous rules*.
Finally, we can use synchronization labels (i.e. labels in system/element net transitions) to enforce the simultaneous execution of two transitions, one with a label a and one with a label $\overline{a}$. Two possibilities are allowed in Nested Petri nets: the synchronization of a transition of the system net and of an element net (*vertical synchronization*), or the simultaneous execution of transitions of two distinct element nets residing in the same system place (*horizontal synchronization*). Notice that vertical synchronization modifies both the marking of the system net and the internal structure of (some) tokens.

***c*P-systems as nested Petri nets** In this section we show that *c*P-systems can be encoded as 2-level nested Petri nets in which the system net is a P/T system and the element net is a P/T system with inhibitor arcs.

Assume a *c*P-system $\Pi = (V, N, R, \mu_0)$. We build a 2-level nested Petri nets as follows. The system net is a P/T system with a places $CONF$ used to contain all conformons in a current configuration of Π, and a place $CREATE_r$ for each creation rule $r \in R$. The transitions of the system net are transport rules that model creation rules used to non-deterministically inject new conformons in the place $CONF$. Namely, for each creation rule $r \in R$ we add a transport rule t_r with pre-set $\{\!\{CREATE_r\}\!\}$ and post-set $\{\!\{CREATE_r, CONF\}\!\}$. We assume here that $CREATE_r$ is initialized with a single element net that models the conformon created by rule r. The transition t_r makes a copy of such an element net and puts it in place $CONF$.

An element net N_c denotes a single conformon c. It is defined by a P/T system with places $P = V \cup N \cup \{E\}$. Only one place of those in N and only one place of those in V can be marked in the same instant. The marked places correspond to the name and current location of c. Furthermore, the number of tokens in place E denotes the current amount of energy of c.

To model an internal rule $r = A \xrightarrow[m]{e} B$ we use a horizontal step between two distinct element nets N_1 and N_2, i.e., a pair $(t_{r,1}, t_{r,2})$ of element net transitions with synchronized labels r and $\overline{r}$ such that:

- ${}^\bullet t_{r,1}$ has one occurrence of A, one of m, and e of E, i.e., it is enabled if N_1 represents a conformon with name A in membrane m and at least e units of energy; those units are subtracted from place E in N_1.
- ${}^\bullet t_{r,2}$ has one occurrence of B and one of m, i.e., it is enabled if N_2 represents a conformon with name B in membrane m.
- $t_{r,1}^\bullet$ has one occurrence of A and one of m.
- $t_{r,2}^\bullet$ has one occurrence of B, one of m, and e of E, i.e., e units of energy are transferred to place E in N_2.

To model a passage rule $r = m \xhookrightarrow{p} n$ with condition $x \geq e$, we use an autonomous step. Specifically, we define an element net transition t_r such that:

- ${}^\bullet t_r$ has one occurence of m, and e occurrences of E, i.e., it is enabled if N_1 is in membrane m and at least e units of energy.
- $t_r^\bullet$ has one occurrences of n, and and e occurrences of E, i.e., N_1 represents now a conformon (with the same name) in membrane n. Its energy is not changed (we first subtract e tokens to check the condition $x \geq e$) and then add e tokens back to place E in N_1).

To model a passage rule r with condition $x = e$, we can add to each transition $t_{r,A}$ with $A \in V$ the test $= e$ on place E. It is easy to define this test by using P/T transitions with inhibitor arcs. Rules with conditions $x \leq e$ for $e > 0$ can be encoded by splitting the test into $x = 0, \ldots, x = e$.

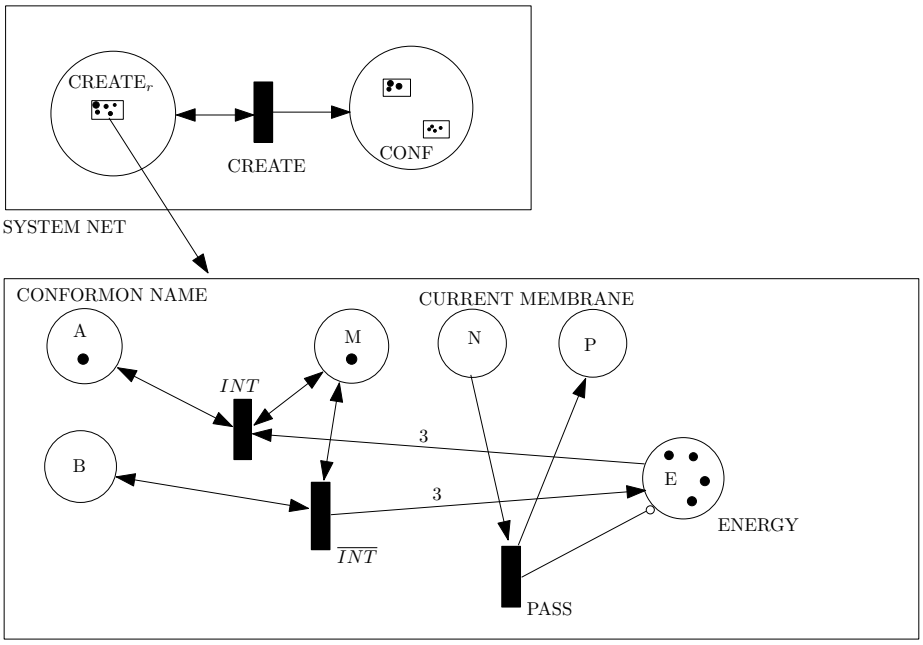


Fig. 4.1 Example of nested Petri net.

Since each element net maintains information about name, value and location the content of place $CONF$ corresponds to the current configuration of Π.

Example 2 Assume a *c*P-system Π with $V = \{A, B\}$, $N = \{M, L, P\}$, the creation rule $r = \overset{4}{\underset{M}{\rightarrow}} A$, the internal rule $INT = A \overset{3}{\underset{M}{\rightarrow}} B$, and the passage rule $PASS = L \overset{p}{\hookrightarrow} P$ with $p(x) \overset{\text{def}}{=} x = 0$. The 2-level nested Petri nets that encodes the *c*P-systems Π is shown in Fig. 4.1. We use here circles to denote places, rectangles to denote transitions, an arrow from a circle to a rectangle to denote places in the pre-set and an arrow from a rectangle to a circle to denote places in the post-sets of transitions; we label arrows with numbers to indicate a multiplicity greater than 1 of a place in the pre-/post-set. An inhibitor arc is represeted by an arrow with a circle. The system net place $CREATE_r$ is used to keep a copy of the conformon $[A, 4]$ so as to non-deterministically inject new ones in the current configuration (i.e. the place $CONF$). The element net has places to model names, membranes, and energy. The internal rule is modelled by the pair of transitions with labels INT and $\overline{INT}$. When executed simultaneously (within the place $CONF$ of the system net) by two distinct element net (one executes INT and the other executes $\overline{INT}$) their effect consists in moving 3 tokens from the place E of an element net with tokens in A, M to the place E of an element net with tokens in the places B, M. Notice that tokens of the element nets are objects with no structure.

The passage rule is modelled by the element net transition with label $PASS$. It simply checks that E is empty with an inhibitor arc and then moves a token from the place N to the place P (it changes the location of the element net). Notice that the system net place $CONF$ may contain an arbitrary number of element nets (the corresponding P/T system is unbounded).

It is important to notice that 2-level nested Petri nets in which element nets have inhibitor arcs are Turing equivalent [9]. This result is consistent with the analysis of the expressive power of full *c*P-systems [5]. From the previous observations, restricted *c*P-systems are a subclass of nested Petri nets in which both the system and the element nets are defined by P/T systems. From the results obtained for well-structured subclasses of nested Petri nets in [9], we obtain an indirect proof for decidability of *coverability* of restricted *c*P-systems.

The connection between *c*P-systems and nested nets can be exploit to extend the model in several ways. As an example, for restricted passage rules, coverability remains decidable when extending *c*P-systems with: conformons defined by a list of pairs name-value instead of a single pair; rules that *transfer* all the energy from A to B; or conformons defined by a state machine (i.e. with an internal state instead of statically assigned type).

4.3 Restricted *c*P-systems vs CMRS Restricted *c*P-systems can also be modelled in CMRS, an extension of Petri nets in which tokens carry natural numbers.

Constrained multiset rewriting systems (CMRS) CMRS [2] are inspired to formulations of colored Petri nets in term rewriting. A token with data d in place p is represented here as a term $p(d)$, a marking as a multiset of terms, and a transition as a (conditional) multiset rewriting rule. More precisely, let *term* be an element $p(x)$ where p belong to a finite set of predicate symbols $\mathbb{P}$ (places) and x is a variable ranging over natural numbers. We often call a term $p(t)$ with $p \in P$ a *p-term* or *P-term*. A element $p(v)$ with $p \in \mathbb{P}$ and $v \in \mathbb{N}_0$ is called a *ground term*.
A configuration is a (finite) multiset of ground terms. A CMRS is a set of rewriting rules with constraints of the form $r = L \rightsquigarrow R : \Psi$ that allows to transform (rewrite) multisets into multisets. More precisely, L and R are multisets of terms (with variables) and Ψ is a (possibly empty) finite conjunction of *gap-order* constraints of the form: $x + c < y$, $x \leq y$, $x = y$, $x < c$, $x > c$, $x = c$ where x, y are variables appearing in L and/or R and $c \in \mathbb{N}_0$ is a constant.
A rule r is enabled at a configuration c if there exists a valuation of the variables Val such that $Val(\Psi)$ is satisfied. Firing r at c leads to a new multiset c', noted $c \xrightarrow{r} c'$, with $c' = c \ominus Val(L) \oplus Val(R)$ where $Val(L)$, resp. $Val(R)$, is the multiset of ground terms obtained from L, resp. R, by replacing each variable x by $Val(x)$.

As an example, consider the CMRS rule:

$$\rho = [p(x)\ ,\ q(y)] \ \rightsquigarrow\ [q(z)\ ,\ r(x)\ ,\ r(w)] \ \ :\ \ \{x + 2 < y\ ,\ x + 4 < z\ ,\ z < w\}$$

A valuation which satisfies the condition is $Val(x) = 1$, $Val(y) = 4$, $Val(z) = 8$, and $Val(w) = 10$. A CMRS configuration is a multisets of ground terms, e.g., $[p(1), p(3), q(4)]$. Therefore, we have that $[p(1), p(3), q(4)] \xrightarrow{\rho} [p(3), q(8), r(1), r(10)]$.

A CMRS is well-structured with respect to the well-quasi ordering $\preceq_c$ defined as follows. Given a configuration c, let $V(c) = \{i \in \mathbb{N}_0 \mid \exists p(i) \in c\}$, and $c_{=i} : \mathbb{P} \mapsto \mathbb{N}_0$ with $i \in \mathbb{N}_0$ be the multi set such that $c_{=i}(p) = c(p_i)$ for any $p \in \mathbb{P}$. Then, we have that $c \preceq_c c'$ iff there exists an injective function $h : V(c) \mapsto \mathbb{N}_0$ such that (*i*) for any $i \in V(c) : c_{=i} \leq c'_{=h(i)}$; (*ii*) for any $i \in V(c)$ s.t. $i \leq cmax : i = h(i)$; (*iii*) for any $i, j \in V(c) \cup \{0\}$ s.t. $i < j$ and $j > cmax : j - i < h(j) - h(i)$. A symbolic algorithm to check coverability – w.r.t. $\preceq_c$ – is described in [2].

Restricted *c*P-systems as CMRS A *c*P-configuration μ is mapped to a CMRS configuration as follows. A conformon $c = [A, x]$ in membrane m is represented by means of a multiset of terms

$$\mathbb{M}^v_{c,m} = [conf_{A,m}(v)] \oplus \mathbb{O}^v_x$$

where $\mathbb{O}^x_v$ is the multiset with x occurrences of the term $u(v)$, i.e.,

$$\mathbb{O}^v_x = [\underbrace{u(v), \ldots, u(v)}_{x-times}]$$

where v is a natural number used as a unique identifier for the conformon c. The u-terms with parameter v are used to count the amount of energy of conformon with identifier v. E.g. if $c = [ATP, 4]$ then $\mathbb{M}^2_{c,m} = [conf_{ATP,m}(2), u(2), u(2), u(2), u(2)]$ – 4 occurrences of $u(2)$ – where 2 is the unique identifier of conformon c. Furthermore, if $c = [ATP, 0]$, then $\mathbb{M}^2_{c,m} = [conf_{ATP,m}(2)]$. Thus, we use $[conf_{A,m}(v)]$ to model a conformon with zero energy and identifier v.

A representation $Rep(\mu)$ of a *c*P-configuration μ is obtained by assigning a distinct indentifier to each conformon and by taking the (multiset) union of the representations of each conformons in μ. Formally, let μ contains r membranes such that $\mu(m_i)$ contains the conformons $c_{1,i}, \ldots, c_{i,n_i}$ for $i : 1, \ldots, r$ and $n_1 + \ldots + n_r = k$, then

$$Rep(\mu)^V = (\bigoplus_{j=1}^{n_1} \mathbb{M}^{v_{1,j}}_{c_{1,j},m_1}) \oplus \ldots \oplus (\bigoplus_{j=1}^{n_r} \mathbb{M}^{v_{r,j}}_{c_{r,j},m_r})$$

where $V = (v_{1,1}, \ldots, v_{1,n_1}, \ldots, v_{r,1}, \ldots, v_{r,n_r})$ are k distinct natural numbers working as identifiers of the k conformons in μ. Identifiers of conformons in the initial configuration μ_0 are non-deterministically chosen at the beginning of the simulation using the following rule:

$$[init] \rightsquigarrow [fresh(v)] \oplus Rep(\mu_0)^V \ : \ \{v_{1,1} < \ldots < v_{1,n_1} < v_{r,1} < v_{r,n_r} < v\}$$

where V is a vector of variables that denotes conformon indentifiers (as described in the def. of $Rep(\mu)^V$). Furthermore, we maintain a fresh identifier v in the $fresh$-term (used to dynamically create other conformons).

The rules of a restricted *c*P-system are simulated via the following CMRS rules working on CMRS representations of configurations.

- Creation of $c = [A, x]$ inside m:

$$[fresh(x)] \ \rightsquigarrow \ [fresh(y)] \oplus \mathbb{M}^{x}_{c,m} \ : \ \{x < y\}$$

  We simply inject a new multiset of terms with parameter x stored in the $fresh$-term and reset the fresh value.
- A and B in membrane m exchange e units of energy:

$$[conf_{A,m}(x), conf_{B,m}(y)] \oplus \mathbb{O}^{x}_{e} \ \rightsquigarrow \ [conf_{A,m}(x), conf_{B,m}(y)] \oplus \mathbb{O}^{y}_{e} \ : \ true$$

  Notice that, by definition of the CMRS operational semantics, the rule is enabled only when there are at least e occurrences of u-terms with parameter x (identifier of A) and where there exists a conformon B with identifier y (x and y are variables ranging over natural numbers). The passage of energy from A (with identifier x) to B (with identifier y) is simply defined by changing the parameter x of e occurrences of u-terms into y.
- Passage rule from membrane m to n conditioned by the predicate $p(x) \stackrel{\text{def}}{=} x \geq c$: For each conformon name A:

$$[conf_{A,m}(x)] \oplus \mathbb{O}^{x}_{c} \ \rightsquigarrow \ [conf_{A,n}(x)] \oplus \mathbb{O}^{x}_{c}$$

  Notice that, by definition of the CMRS operational semantics, the rule is enabled only when there are at least c occurrences of u-terms with parameter x (identifier of A). The current location of A is stored in the term $conf_{A,m}(x)$. The passage to membrane n is defined by changing the term $conf_{A,m}(x)$ into $conf_{A,n}(x)$. The u-terms with the same parameter are not consumed (i.e. they occur both in the left-hand side and in the right-hand side of the rule).

From the results obtained for CMRS [2], we obtain another indirect proof for decidability of *coverability* of restricted *c*P-systems. The connection between *c*P-systems and CMRS can be used to devise extensions of the conformon model in which, e.g., conformon have different priorities or ordered with respect to some other parameter. This can be achieved by ordering the parameters of the multiset of terms used to encode each conformon. CMRS rules can deal with such an ordering by using conditions on parameters of terms in a rule of the form $x < y$.

5 Related Work and Conclusions

In the paper we have investigated the decidability of computational properties of conformon P-systems like reachability and coverability. More specifically, we have shown that, although undecidable for the full model, the coverability problem is decidable for a fragment with restricted types of predicates in passage rules.

To our knowledge, this is the first work devoted to the qualitative analysis of conformon P-systems, and to the comparison with other models like nested Petri nets and CMRS. The expressiveness of the conformon P-systems is studied in [5] by using a reduction to counter machines with zero test (Turing equivalent). We use such a result to show that coverability is undecidable for the full model. The decidability or reachability for the full model is not in contrast with its great expressive power. Indeed, in the reachability problem the target configuration contains precise information about the history of the computation, e.g., the total amount of energy exchanged during the computation. These information cannot be expressed in the coverability problem, where we can only fix part of the information of target configurations. In this sense, coverability seems a better measure for the expressiveness of this kind of computational models.

In the paper we have compared this result with similar results obtained for other models like nested Petri nets and constrained multiset rewriting systems. The direct proof presented in the paper and the corresponding algorithm can be viewed however as a first step towards the development of automated verification tools for biologically inspired models. The kind of qualitative analysis that can be performed using our algorithm is complementary to the simulation techniques used in quantitative analysis of natural and biological systems. Indeed, in qualitative analysis we consider all possible executions with no probability distributions on transitions, whereas in quantitative analysis one often considers a single simulation by associating probabilities to each single transitions. Unfortunately, the rates of reactions are often unknown and, thus, extrapolated from known data to make the simulation feasible. Qualitative analysis requires instead only the knowledge of the dynamics of a given natural model. Automated verification methods can thus be useful to individuate structural properties of biological models.

Acknowledgements. Research fellow supported by the Belgian National Science Foundation.

Bibliography

[1] P. A. Abdulla, K. Čerāns, B. Jonsson, and T. Yih-Kuen. General decidability theorems for infinite-state systems. LICS '96: 313–321.
[2] P. A. Abdulla and G. Delzanno: On the Coverability Problem for Constrained Multiset Rewriting Systems. AVIS '06 (ETAPS-workshop). Available in the authors' home page.
[3] Z. Dang, O. Ibarra, C. Li, G. Xie. On Model Checking of P Systems. UC 2005: 82-93.
[4] A. Finkel, Ph. Schnoebelen. Well-structured transition systems everywhere! TCS 256(1-2), 2001: 63-92.
[5] P. Frisco. The conformon-P system: a molecular and cell biology-inspired computability model. TCS 312(2-3), 2004: 295-319.

[6] P.Frisco and D. W. Corne. Dynamics of HIV infection studied with Cellular Automata and conformon-P systems, BioSystems 91 (3), 2008: 531-544.
[7] G. Higman. Ordering by divisibility in abstract algebras. London Math. Soc. (3), 2(7), 1952: 326336.
[8] I. A. Lomazova. Nested petri nets. Fundamenta Informaticae 43(1-4), 2000: 195 - 214.
[9] I. A. Lomazova and Ph. Schnoebelen. Some Decidability Results for Nested Petri Nets. Ershov Mem. Conf. 1999: 208-220.
[10] G. Paun. Computing with membranes. JCSS 61(1), 2000: 108-143.
[11] W. Reisig. Petri Nets: An Introduction. Springer, 1985.

Dual P Systems

Oana Agrigoroaiei[1] and Gabriel Ciobanu[1,2]

[1]Romanian Academy, Institute of Computer Science
Blvd. Carol I no.8, 700505 Iaşi, Romania
oanaag@iit.tuiasi.ro

[2]"A.I.Cuza" University of Iaşi, Faculty of Computer Science
Blvd. Carol I no.11, 700506 Iaşi, Romania
gabriel@info.uaic.ro

This paper aims to answer the following question: given a P system configuration M, how do we find each configuration N such that N evolves to M in one step? While easy to state, the problem has not a simple answer. To provide a solution to this problem for a general class of P systems with simple communication rules and without dissolution, we introduce the dual P systems. Essentially these systems reverse the rules of the initial P system and find N by applying reversely valid multisets of rules. We prove that in this way we find exactly those configurations N which evolve to M in one step.

1 Introduction

Often when solving a (mathematical) problem, one starts from the end and tries to reach the hypothesis. P systems [4] are often used to solve problems, so finding a method which allows us to go backwards is of interest. When looking at a cell-like P system with rules which only involve object rewriting (of type $u \to v$, where u, v are multisets of objects) in order to reverse a computation it is natural to reverse the rules ($u \to v$ becomes $v \to u$) and find a condition equivalent to maximal parallelism. The dual P system $\widetilde{\Pi}$ is the one with the same membranes as Π and the rules of Π reversed. However, when rules of type $u \to (v, out)$ or $u \to (v, in_{child})$ are used, two ways of reversing computation appear. The one we focus on is to employ a special type of rule reversal and to move the rules between membranes: for example, $u \to (v, out)$ associated to the membrane with label i in Π is replaced with $v \to (u, in_i)$ associated to the membrane with label $parent(i)$ in $\widetilde{\Pi}$. This is described in detail in Section 4. Another way of defining the dual P system is by reversing all the rules without moving them between membranes (and thus allow rules of form $(v, out) \to u$). To capture the backwards computation we have to move objects according to the existence of communicating rules in the P system. The object movement corresponds to reversing the message sending stage of the evolution of a membrane. After that the maximally parallel rewriting stage is reversed. This is only sketched in Section 5 as a starting point for further research.

The structure μ of a P system is represented by a tree structure (with the $skin$ as its root), or equivalently, by a string of correctly matching parentheses, placed in a unique pair of matching parentheses; each pair of matching parentheses corresponds to a membrane. Graphically, a membrane structure is represented by a Venn diagram in which two sets can be either disjoint, or one a subset of the other. The membranes are labelled in a one-to-one manner. A membrane without any other membrane inside is said to be elementary.

A *membrane system* of degree m is a tuple $\Pi = (O, \mu, w_1, \ldots w_m, R_1, \ldots, R_m, i_o)$ where:

- O is an alphabet of objects;
- μ is a membrane structure, with the membranes labelled by natural numbers $1, \ldots, m$, in a one-to-one manner;
- w_i are multisets over O associated with the regions $1, \ldots, m$ defined by μ;
- $R_1, \ldots, R_m$ are finite sets of rules associated with the membranes with labels $1, \ldots, m$; the rules have the form $u \rightarrow v$, where u is a non-empty multiset of objects and v a multiset over messages of the form $(a, here), (a, out), (a, in_j)$;

The membrane structure μ and the multisets of objects and messages from its compartments define a *intermediate configuration* of a P system. If the multisets from its compartments contain only objects, they define a *configuration*. For a intermediate configuration M we denote by $w_i(M)$ the multiset contained in the inner membrane with label i. We denote by $\mathcal{C}^{\#}(\Pi)$ the set of intermediate configurations and by $\mathcal{C}(\Pi)$ the set of configurations of the P system Π.

Since we work with two P systems at once (namely Π and $\widetilde{\Pi}$), we use the notation $R_1^{\Pi}, \ldots, R_m^{\Pi}$ for the sets of rules $R_1, \ldots, R_m$ of the P system Π.

We consider a multiset w over a set S to be a function $w : S \rightarrow \mathbb{N}$. When describing a multiset characterised by, for example, $w(s) = 1, w(t) = 2$, $w(s') = 0, s' \in S \backslash \{s, t\}$, we use its string representation $s + 2t$, to simplify its description. To each multiset w we associate its support, denoted by $supp(w)$, which contains those elements of S which have a non-zero image. A multiset is called non-empty if it has non-empty support. We denote the empty multiset by 0_S. The sum of two multisets w, w' over S is the multiset $w + w' : S \rightarrow \mathbb{N}, (w + w')(s) = w(s) + w'(s)$. For two multisets w, w' over S we say that w is contained in w' if $w(s) \leq w'(s), \forall s \in S$. We denote this by $w \leq w'$. If $w \leq w'$ we can define $w' - w$ by $(w' - w)(s) = w'(s) - w(s)$. To work in a uniform manner, we consider all multisets of objects and messages to be over

$$\Omega = O \cup O \times \{out\} \cup O \times \{in_j \mid j \in \{1, \ldots, m\}\}$$

Definition 1 *The set $\mathcal{M}(\Pi)$ of membranes in a P system Π together with the membrane structure are inductively defined as follows:*

- *if i is a label and w is a multiset over $O \cup O \times \{out\}$ then $\langle i|w\rangle \in \mathcal{M}(\Pi)$; $\langle i|w\rangle$ is called an* elementary membrane*, and its structure is $\langle\rangle$;*
- *if i is a label, $M_1, \ldots, M_n \in \mathcal{M}(\Pi), n \geq 1$ have distinct labels $i_1, \ldots, i_n$, each M_k has structure μ_k and w is a multiset over $O \cup O \times \{out\} \cup O \times \{in_{i_1}, \ldots, in_{i_n}\}$ then $\langle i|w; M_1, \ldots, M_n\rangle \in \mathcal{M}(\Pi)$; $\langle i|w; M_1, \ldots, M_n\rangle$ is called a* composite membrane*, and its structure is $\langle \mu_1 \ldots \mu_n\rangle$.*

Note that if i is the label of the skin membrane then $\langle i|w; M_1, \ldots, M_n\rangle$ defines an intermediate configuration.

We use the notations $parent(i)$ for the label indicating the parent of the membrane labelled by i (if it exists) and $children(i)$ for the set of labels indicating the children of the membrane labelled by i, which can be empty.

By *simple* communication rules we understand that all rules inside membranes are of the form $u \to v$ where u is a multiset of objects ($supp(u) \subseteq O$) and v is either a multiset of objects, or a multiset of objects with the message in_j ($supp(v) \subseteq O \times \{in_j\}$ for a $j \in \{1, \ldots, m\}$) or a multiset of objects with the message out ($supp(v) \subseteq O \times \{out\}$). Moreover we suppose that the $skin$ membrane does not have any rules involving objects with the message out.

We use multisets of rules $\mathcal{R} : R_i^\Pi \to \mathbb{N}$ to describe maximally parallel application of rules. For a rule $r = u \to v$ we use the notations $lhs(r) = u, rhs(r) = v$. Similarly, for a multiset $\mathcal{R}$ of rules from R_i^Π, we define the following multisets over Ω:

$$lhs(\mathcal{R})(o) = \sum_{r \in R_i^\Pi} \mathcal{R}(r) \cdot lhs(r)(o) \text{ and } rhs(\mathcal{R})(o) = \sum_{r \in R_i^\Pi} \mathcal{R}(r) \cdot rhs(r)(o)$$

for each object or message $o \in \Omega$. The following definition captures the meaning of "maximally parallel application of rules":

Definition 2 *We say that a multiset of rules $\mathcal{R} : R_i^\Pi \to \mathbb{N}$ is* valid *in the multiset w if $lhs(\mathcal{R}) \leq w$. The multiset $\mathcal{R}$ is called* maximally valid *in w if it is valid in w and there is no rule $r \in R_i^\Pi$ such that $lhs(r) \leq w - lhs(\mathcal{R})$.*

2 P Systems with One Membrane

Suppose that the P system Π consists only of the $skin$ membrane, labelled by 1. Since the membrane has no children and we have assumed it has no rules concerning out messages, all its rules are of form $u \to v$, with $supp(u), supp(v) \subseteq O$. Given the configuration M in the system $\Pi = (O, \mu, w_1, R_1^\Pi)$ we want to find all configurations

N such that N rewrites to M in a single maximally parallel rewriting step. To do this we define the dual P system $\widetilde{\Pi} = (O, \mu, w_1, R_1^{\widetilde{\Pi}})$, with evolution rules given by:

$$(u \to v) \in R_1^{\widetilde{\Pi}} \text{ if and only if } (v \to u) \in R_1^{\Pi}$$

For each $M = \langle 1|w\rangle \in \mathcal{C}^{\#}(\Pi)$, we consider the dual intermediate configuration $\widetilde{M} = \langle 1|w\rangle \in \mathcal{C}^{\#}(\widetilde{\Pi})$ which has the same content ($w = w_1(\widetilde{M}) = w_1(M)$) and membrane structure as M. Note that the dual of a configuration is a configuration. The notation $\widetilde{M}$ is used to emphasize that it is an intermediate configuration of the system $\widetilde{\Pi}$.

The name *dual* is used for the P system $\widetilde{\Pi}$ under the influence of category theory, where the dual category is the one obtained by reversing all arrows.

Remark 1 *Note that using the term of* dual *for* $\widetilde{\Pi}$ *is appropriate because* $\widetilde{\widetilde{\Pi}} = \Pi$.

When we reverse the rules of a P system, dualising the maximally parallel application of rules requires a different concept than the *maximal validity* of a multiset of rules.

Definition 3 *The multiset* $\mathcal{R} : R_i^{\Pi} \to \mathbb{N}$ *is called* reversely valid *in the multiset* w *if it is valid in* w *and there is no rule* $r \in R_i^{\Pi}$ *such that* $rhs(r) \leq w - lhs(\mathcal{R})$.

Note that the difference from *maximally valid* is that here we use the right-hand side of a rule r in $rhs(r) \leq w - lhs(\mathcal{R})$, instead of the left-hand side.

Example 1 Consider the configuration $M = \langle 1|b + c\rangle$, in the P system $\widetilde{\Pi}$ with $O = \{a, b, c\}$, $\mu = \langle\rangle$ and with evolution rules $R_1^{\Pi} = \{r_1, r_2\}$, where $r_1 = a \to b$, $r_2 = b \to c$. Then $\widetilde{M} = \langle 1|b + c\rangle \in \mathcal{C}(\widetilde{\Pi})$, with evolution rules $R_1^{\widetilde{\Pi}} = \{\widetilde{r_1}, \widetilde{r_2}\}$, where $\widetilde{r_1} = b \to a$, $\widetilde{r_2} = c \to b$. The valid multisets of rules in $w_1(\widetilde{M}) = b + c$ are $0_{R_1^{\widetilde{\Pi}}}, \widetilde{r_1}, \widetilde{r_2}$ and $\widetilde{r_1} + \widetilde{r_2}$. The reversely valid multiset of rules $\widetilde{\mathcal{R}}$ in $w(\widetilde{M_1})$ can be either $\widetilde{r_1}$ or $\widetilde{r_1} + \widetilde{r_2}$. If $\widetilde{\mathcal{R}} = \widetilde{r_1}$ then $\widetilde{M}$ rewrites to $\langle 1|a + c\rangle$; if $\widetilde{\mathcal{R}} = \widetilde{r_1} + \widetilde{r_2}$ then $\widetilde{M}$ rewrites to $\langle 1|a + b\rangle$. These yield the only two configurations that can evolve to M in one maximally parallel rewriting step (in Π). This example clarifies why reversely valid multisets of rules must be applied: validity ensures that some objects are consumed by rules $\widetilde{r}$ (dually, they were produced by some rules r) and reverse validity ensures that objects like b (appearing in both the left and right-hand sides of rules) are always consumed by rules $\widetilde{r}$ (dually, they were surely produced by some rules r, otherwise it would contradict maximal parallelism for the multiset $\mathcal{R}$).

Note that if $M' = \langle 1 \mid 2a\rangle$ in the P system Π, then there is no multiset of rules $\widetilde{\mathcal{R}}$ valid in $w_1(\widetilde{M'}) = 2a$ for the dual $\widetilde{M'}$. This happens exactly because there is no configuration N' such that N' rewrites to M' by applying at least one of the rules r_1, r_2.

We present the operational semantics for both maximally parallel application of rules (mpr) and inverse maximally parallel application of rules ($\widetilde{mpr}$) on configurations in a P system with one membrane.

Definition 4

- $\langle 1|w\rangle \xrightarrow{\mathcal{R}}_{mpr} \langle 1|w - lhs(\mathcal{R}) + rhs(\mathcal{R})\rangle$ *if and only if* $\mathcal{R}$ *is maximally valid in* w*;*
- $\langle 1|w\rangle \xrightarrow{\mathcal{R}}_{\widetilde{mpr}} \langle 1|w - lhs(\mathcal{R}) + rhs(\mathcal{R})\rangle$ *if and only if* $\mathcal{R}$ *is reversely valid in* w*.*

The difference between the two semantics is coming from the difference between the conditions imposed on the multiset $\mathcal{R}$ (maximally valid and reversely valid, respectively).

For a multiset $\mathcal{R}$ of rules over R_1^{Π} we denote by $\widetilde{\mathcal{R}}$ the multiset of rules over $R_1^{\widetilde{\Pi}}$ for which $\widetilde{\mathcal{R}}(u \to v) = \mathcal{R}(v \to u)$. Then $lhs(\mathcal{R}) = rhs(\widetilde{\mathcal{R}})$ and $rhs(\mathcal{R}) = lhs(\widetilde{\mathcal{R}})$.

Proposition 1 $N \xrightarrow{\mathcal{R}}_{mpr} M$ *if and only if* $\widetilde{M} \xrightarrow{\widetilde{\mathcal{R}}}_{\widetilde{mpr}} \widetilde{N}$.

Proof If $N \xrightarrow{\mathcal{R}}_{mpr} M$ then $\mathcal{R}$ is maximally valid in $w_1(N)$ and $w_1(M) = w_1(N) - lhs(\mathcal{R}) + rhs(\mathcal{R})$; then $w_1(M) - rhs(\mathcal{R}) = w_1(N) - lhs(\mathcal{R})$. By duality, we have $w_1(M) = w_1(\widetilde{M})$ and $rhs(\mathcal{R}) = lhs(\widetilde{\mathcal{R}})$; it follows that $w_1(\widetilde{M}) - lhs(\widetilde{\mathcal{R}}) = w_1(N) - lhs(\mathcal{R}) \geq 0$, therefore $lhs(\widetilde{\mathcal{R}}) \leq w_1(\widetilde{M})$, and so $\widetilde{\mathcal{R}}$ is valid in $\widetilde{M}$. Suppose $\widetilde{\mathcal{R}}$ is not reversely valid in $w_1(\widetilde{M})$, i.e. there exists $\widetilde{r} \in R_1^{\widetilde{\Pi}}$ such that $rhs(\widetilde{r}) \leq w_1(\widetilde{M}) - lhs(\widetilde{\mathcal{R}})$, which is equivalent to $lhs(r) \leq w_1(M) - rhs(\mathcal{R})$. Since $w_1(M) - rhs(\mathcal{R}) = w_1(N) - lhs(\mathcal{R})$ it follows that $\mathcal{R}$ is not maximally valid in $w_1(N)$, which yields a contradiction.

If $\widetilde{M} \xrightarrow{\widetilde{\mathcal{R}}}_{\widetilde{mpr}} \widetilde{N}$ then $\widetilde{\mathcal{R}}$ is reversely valid in $w_1(\widetilde{M})$; since $w_1(N) - lhs(\mathcal{R}) = w_1(\widetilde{M}) - lhs(\widetilde{\mathcal{R}}) \geq 0$ it follows that $\mathcal{R}$ is valid in $w_1(N)$. If we suppose that $\mathcal{R}$ is not maximally valid in $w_1(N)$ then, reasoning as above, we obtain that $\widetilde{\mathcal{R}}$ is not reversely valid in $w_1(\widetilde{M})$(contradiction). □ □

3 P Systems without Communication Rules

If the P system has more than one membrane but it has no communication rules (i.e. no rules of form $u \to v$, with $supp(v) \subseteq O \times \{out\}$ or $supp(v) \subseteq O \times \{in_j\}$) the method of reversing the computation is similar to that described in the previous section. We describe it again but in a different way, since here we introduce the notion of a (valid) system of multisets of rules for a P system Π. This notion is useful for P systems without communication rules, and is fundamental in reversing the computation of a P

system with communication rules. This section provides a technical step from Section 2 to Section 4.

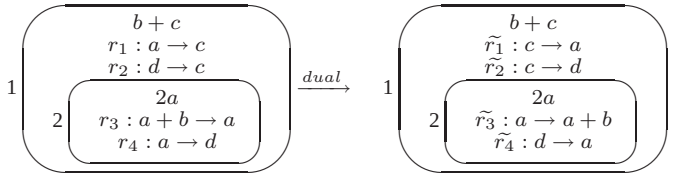
Definition 5 *A* system *of multisets of rules for a P system* Π *of degree* m *is a tuple* $\mathcal{R} = (\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_m)$*, where each* $\mathcal{R}_i$ *is a multiset over* R_i^{Π}*,* $i \in \{1, \ldots, m\}$.

A system of multisets of rules $\mathcal{R}$ is called *valid*, *maximally valid* or *reversely valid* in the configuration M if each $\mathcal{R}_i$ is valid, maximally valid or reversely valid in the multiset $w_i(M)$, which, we recall, is the multiset contained in the inner membrane of configuration M which has label i.

The P system $\widetilde{\Pi}$ dual to the P system Π is defined analogously to the one in Section 2: $\widetilde{\Pi} = (O, \mu, w_1, \ldots w_m, R_1^{\widetilde{\Pi}}, \ldots, R_m^{\widetilde{\Pi}})$ where $(u \to v) \in R_1^{\widetilde{\Pi}}$ if and only if $(v \to u) \in R_1^{\Pi}$. Note that $\widetilde{\widetilde{\Pi}} = \Pi$.

If $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_m)$ is a system of multisets of rules for a P system Π, we denote by $\widetilde{\mathcal{R}}$ the system of multisets of rules for the dual P system $\widetilde{\Pi}$ given by $\widetilde{\mathcal{R}} = (\widetilde{\mathcal{R}_1}, \ldots, \widetilde{\mathcal{R}_2})$.

Example 2 Consider the configuration $M = \langle 1|b + c; N\rangle$, $N = \langle 2|2a\rangle$ of the P system Π with evolution rules $R_1^{\Pi} = \{r_1, r_2\}$, $R_2^{\Pi} = \{r_3, r_4\}$, where $r_1 = a \to c$, $r_2 = d \to c$, $r_3 = a + b \to a$, $r_4 = a \to d$. Then $\widetilde{M} = \langle 1|b + c; \langle 2|2a\rangle\rangle$, with evolution rules $R_1^{\widetilde{\Pi}} = \{\widetilde{r_1}, \widetilde{r_2}\}$, $R_2^{\widetilde{\Pi}} = \{\widetilde{r_3}, \widetilde{r_4}\}$, where $\widetilde{r_1} = c \to a$, $\widetilde{r_2} = c \to d$, $\widetilde{r_3} = a \to a + b$, $\widetilde{r_4} = d \to a$. In order to find all membranes which evolve to M in one step, we look for a system $\widetilde{\mathcal{R}} = (\widetilde{\mathcal{R}_1}, \widetilde{\mathcal{R}_2})$ of multisets of rules, which is reversely valid in the configuration $\widetilde{M}$. Then $\widetilde{\mathcal{R}_1}$ can be either $0_{R_1^{\widetilde{\Pi}}}$, $\widetilde{r_1}$ or $\widetilde{r_2}$ and the only possibility for $\widetilde{\mathcal{R}_2}$ is $2\widetilde{r_3}$. We apply $\widetilde{\mathcal{R}}$ to the $skin$ membrane $\widetilde{M}$ and we obtain three possible configurations $\widetilde{P}$ such that $P \Rightarrow M$; namely, P can be either $\langle 1|b + c; \langle 2|2a + 2b\rangle\rangle$ or $\langle 1|b + a; \langle 2|2a + 2b\rangle\rangle$ or $\langle 1|b + d; \langle 2|2a + 2b\rangle\rangle$.



We give a definition of the operational semantics for both maximally parallel application of rules (mpr) and inverse maximally parallel application of rules ($\widetilde{mpr}$) in a P system without communication rules. We use $\mathcal{R}$ as label to suggest that rule application is done simultaneously in all membranes, and thus to prepare the way toward the general case of P systems with communication rules.

Definition 6 *For $M, N \in \mathcal{C}(\Pi)$ we define:*

- $M \xrightarrow{\mathcal{R}}_{mpr} N$ *if and only if* $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_m)$ *is maximally valid in* M *and* $w_i(N) = w_i(M) - lhs(\mathcal{R}_i) + rhs(\mathcal{R}_i)$*;*
- $M \xrightarrow{\mathcal{R}}_{\widetilde{mpr}} N$ *if and only if* $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_m)$ *is reversely valid in* M *and* $w_i(N) = w_i(M) - lhs(\mathcal{R}_i) + rhs(\mathcal{R}_i)$.

The two operational semantics are similar in their effect on the membranes, but differ in the conditions required for the multisets of rules $\mathcal{R}$.

Proposition 2 *If $N \in \mathcal{C}(\Pi)$, then*

$$N \xrightarrow{\mathcal{R}}_{mpr} M \text{ if and only if } \widetilde{M} \xrightarrow{\widetilde{\mathcal{R}}}_{\widetilde{mpr}} \widetilde{N}$$

Proof If $N \xrightarrow{\mathcal{R}}_{mpr} M$ then $\mathcal{R}$ is maximally valid in the configuration N, which means that $\mathcal{R}_i$ is maximally valid in $w_i(N)$, and $w_i(M) = w_i(N) - lhs(\mathcal{R}_i) + rhs(\mathcal{R}_i)$. By using the same reasoning as in the proof of Proposition 1 it follows that $\widetilde{\mathcal{R}_i}$ is reversely valid in $w_i(\widetilde{M})$, for all $i \in \{1, \ldots, m\}$. Therefore $\widetilde{\mathcal{R}}$ is reversely valid in the configuration $\widetilde{M}$ of the dual P system $\widetilde{\Pi}$. Moreover, we have $w_i(\widetilde{N}) = w_i(\widetilde{M}) - lhs(\widetilde{\mathcal{R}_i}) + rhs(\widetilde{\mathcal{R}_i})$, so $\widetilde{M} \xrightarrow{\widetilde{\mathcal{R}}}_{\widetilde{mpr}} \widetilde{N}$.

If $\widetilde{M} \xrightarrow{\widetilde{\mathcal{R}}}_{\widetilde{mpr}} \widetilde{N}$ the proof follows in the same manner. □ □

4 P Systems with Communication Rules

When the P system has communication rules we no longer can simply reverse the rules and obtain a reverse computation; we also have to move the rules between membranes. When saying that we move the rules we understand that the dual system can have rules $\widetilde{r}$ associated to a membrane with label i while r is associated to a membrane with label j (j is either the parent or the child of i, depending on the form of r). We need a few notations before we start explaining in detail the movement of rules.

If u is a multiset of objects ($supp(u) \subseteq O$) we denote by (u, out) the multiset with $supp(u, out) \subseteq O \times \{out\}$ given by $(u, out)(a, out) = u(a)$, for all $a \in O$. More explicitly, (u, out) has only messages of form (a, out), and their number is that of the objects a in u. Given a label j, we define (u, in_j) similarly: $supp(u, in_j) \subseteq O \times \{in_j\}$ and $(u, in_j)(a, in_j) = u(a)$, for all $a \in O$.

The P system $\widetilde{\Pi}$ dual to the P system Π is defined differently from the case of P systems without communication rules: $\widetilde{\Pi} = (O, \mu, w_1, \ldots w_m, R_1^{\widetilde{\Pi}}, \ldots, R_m^{\widetilde{\Pi}})$ such that:

1. $\widetilde{r} = u \to v \in R_i^{\widetilde{\Pi}}$ if and only if $r = v \to u \in R_i^{\Pi}$;
2. $\widetilde{r} = u \to (v, out) \in R_i^{\widetilde{\Pi}}$ if and only if $r = v \to (u, in_i) \in R_{parent(i)}^{\Pi}$;
3. $\widetilde{r} = u \to (v, in_j) \in R_i^{\widetilde{\Pi}}$ if and only if $r = v \to (u, out) \in R_j^{\Pi}$, $i = parent(j)$;

where u, v are multisets of objects. Note the difference between rule duality when there are no communication rules and the current class of P systems with communication rules.

Proposition 3 *The dual of the dual of a P system is the initial P system:*

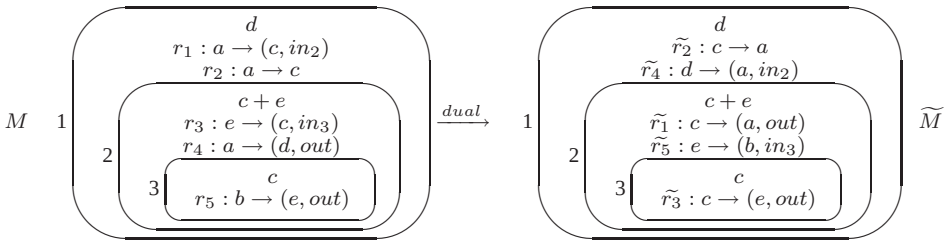
$$\widetilde{\widetilde{\Pi}} = \Pi$$

Proof Clearly, $u \to v \in R_i^{\widetilde{\widetilde{\Pi}}}$ iff $u \to v \in R_i^{\Pi}$. Moreover, $\widetilde{\widetilde{r}} = u \to (v, out) \in R_i^{\widetilde{\widetilde{\Pi}}}$ iff $\widetilde{r} = v \to (u, in_i) \in R_{parent(i)}^{\widetilde{\Pi}}$ which happens iff $r = u \to (v, out) \in R_i^{\Pi}$ (the condition related to the parent amounts to $parent(i) = parent(i)$). Then, $\widetilde{\widetilde{r}} = u \to (v, in_j) \in R_i^{\widetilde{\widetilde{\Pi}}}$ iff $\widetilde{r} = v \to (u, out) \in R_j^{\widetilde{\Pi}}$ and $i = parent(j)$, which happens iff $r = u \to (v, in_j) \in R_{parent(j)=i}^{\Pi}$. □ □

If $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_m)$ is a system of multisets of rules for a P system Π we also need a different dualisation for it. Namely, we denote by $\widetilde{\mathcal{R}}$ the system of multisets of rules for the dual P system $\widetilde{\Pi}$ given by $\widetilde{\mathcal{R}} = (\widetilde{\mathcal{R}_1}, \ldots, \widetilde{\mathcal{R}_2})$, such that:

- if $\widetilde{r} = u \to v \in R_i^{\widetilde{\Pi}}$ then $\widetilde{\mathcal{R}_i}(\widetilde{r}) = \mathcal{R}_i(r)$;
- if $\widetilde{r} = u \to (v, out) \in R_i^{\widetilde{\Pi}}$ then $\widetilde{\mathcal{R}_i}(\widetilde{r}) = \mathcal{R}_{parent(i)}(r)$;
- if $\widetilde{r} = u \to (v, in_j) \in R_i^{\widetilde{\Pi}}$ then $\widetilde{\mathcal{R}_i}(\widetilde{r}) = \mathcal{R}_j(r)$.

Example 3 Consider $M = \langle 1|d; N\rangle$, $N = \langle 2|c + e; P\rangle$, $P = \langle 3|c\rangle$ in the P system Π with $R_1^{\Pi} = \{r_1, r_2\}$, $R_2^{\Pi} = \{r_3, r_4\}$ and $R_3^{\Pi} = \{r_5\}$, where $r_1 = a \to (c, in_2)$, $r_2 = a \to c$, $r_3 = e \to (c, in_3)$, $r_4 = a \to (d, out)$ and $r_5 = b \to (e, out)$. Then $\widetilde{M} = \langle 1|d; \langle 2|c + e; \langle 3|c\rangle\rangle\rangle$ in the dual P system $\widetilde{\Pi}$, with $R_1^{\widetilde{\Pi}} = \{\widetilde{r_2}, \widetilde{r_4}\}$, $R_1^{\widetilde{\Pi}} = \{\widetilde{r_1}, \widetilde{r_5}\}$, $R_3^{\widetilde{\Pi}} = \{\widetilde{r_3}\}$, where $\widetilde{r_1} = c \to (a, out)$, $\widetilde{r_2} = c \to a$, $\widetilde{r_3} = c \to (e, out)$, $\widetilde{r_4} = d \to (a, in_2)$ and $\widetilde{r_5} = e \to (b, in_3)$. For a system of multisets of rules $\mathcal{R} = (r_1 + r_2, 2r_4, 3r_5)$ in Π the dual is $\widetilde{\mathcal{R}} = (2\widetilde{r_4} + \widetilde{r_2}, \widetilde{r_1} + 3\widetilde{r_5}, 0_{R_3^{\widetilde{\Pi}}})$.



The definitions for validity and maximal validity of a system of multisets of rules are the same as in Section 3. However, we need to extend the definition of reverse validity to describe situations arising from a rule being moved.

Definition 7 *A system of multisets of rules $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_n)$ for a P system Π is called* reversely valid *in the configuration M if:*

- *$\mathcal{R}$ is valid in the configuration M (i.e. $lhs(\mathcal{R}_i) \leq w_i(M)$);*
- *$\forall i \in \{1, \ldots, m\}$, there is no rule $r = u \to v \in R_i^\Pi$ such that $rhs(r) = v \leq w_i(M) - lhs(\mathcal{R}_i)$;*
- *$\forall i \in \{1, \ldots, m\}$ such that there exists $parent(i)$, there is no rule $r = u \to (v, in_i) \in R_{parent(i)}^\Pi$ such that $v \leq w_i(M) - lhs(\mathcal{R}_i)$;*
- *$\forall i, j \in \{1, \ldots, m\}$ such that $parent(j) = i$, there is no rule $r = u \to (v, out) \in R_j^\Pi$ such that $v \leq w_i(M) - lhs(\mathcal{R}_i)$.*

While this definition is more complicated than the one in Section 3, it can be seen in the proof of Proposition 4 that it is exactly what is required to reverse a computation in which a maximally parallel rewriting takes place.

Example 3 continued. We look for $\widetilde{\mathcal{R}}$ reversely valid in $\widetilde{M}$. Since $\widetilde{\mathcal{R}}$ must be valid, $\widetilde{\mathcal{R}_1}$ can be equal to $0_{R_1^{\tilde{\Pi}}}$ or $\widetilde{r_4}$; $\widetilde{\mathcal{R}_2}$ equal to $0_{R_2^{\tilde{\Pi}}}$, $\widetilde{r_1}$, $\widetilde{r_5}$ or $\widetilde{r_1} + \widetilde{r_5}$; $\widetilde{\mathcal{R}_3}$ equal to $0_{R_3^{\tilde{\Pi}}}$ or $\widetilde{r_3}$. According to Definition 7, we can look at any of those possibilities for $\mathcal{R}_i$ to see if it can be a component of a reversely valid system $\mathcal{R}$. In this example the only problem (with respect to reverse validity) appears when $\widetilde{\mathcal{R}_2} = 0_{R_2^{\tilde{\Pi}}}$ or when $\widetilde{\mathcal{R}_2} = \widetilde{r_1}$, since in both cases we have $e \leq w_2(\widetilde{M}) - lhs(\widetilde{\mathcal{R}_2})$ and rule $c \to (e, out) \in R_3^{\tilde{\Pi}}$. Let us see why we exclude exactly these two cases. Suppose $\widetilde{\mathcal{R}_2} = \widetilde{r_1}$ and, for example, $\widetilde{\mathcal{R}_1} = \widetilde{r_4}$, $\widetilde{\mathcal{R}_3} = \widetilde{r_3}$. If $\widetilde{\mathcal{R}}$ is applied, $\widetilde{M}$ rewrites to $\langle 1|(a, in_2); \langle 2|(a, out) + e; \langle 3|(e, out)\rangle\rangle\rangle$; after message sending, we obtain $\langle 1|a; \langle 2|a + 2e; \langle 3|0_O\rangle\rangle\rangle$ which cannot rewrite to M while respecting maximal parallelism (otherwise there would appear two c's in the membrane P with label 3). The same thing would happen when $\widetilde{\mathcal{R}_2} = 0_{R_2^{\tilde{\Pi}}}$.

In P systems with communication rules we work with both rewriting and message sending. We have presented two semantics for rewriting in Section 3: $\rightarrow_{mpr}$ (maximally parallel rewriting) and $\rightarrow_{\widetilde{mpr}}$ (inverse maximally parallel rewriting). They are also used here, with the remark that the notion of *reversely valid system* has been extended (see Definition 7).

Before giving the operational semantics for message sending we present a few more notations. Given a multiset $w : \Omega \rightarrow \mathbb{N}$ we define the multisets $obj(w)$, $out(w)$, $in_j(w)$ which consist only of objects (i.e. $supp(obj(w))$, $supp(out(w))$, $supp(in_j(w)) \subseteq O$), as follows:

- $obj(w)$ contains all the objects from w: $obj(w)(a) = w(a), \forall a \in O$;
- $out(w)$ contains all the objects a which are part of a message (a, out) in w: $out(w)(a) = w(a, out), \forall a \in O$;
- $in_j(w)$ contains all the objects a which are part of a message (a, in_j) in w: $in_j(w)$ $(a) = w(a, in_j), \forall a \in O, \forall j \in \{1, \ldots, m\}$.

Definition 8 *For a intermediate configuration M, $M \rightarrow_{msg} N$ if and only if*

$$w_i(N) = obj(w_i(M)) + in_i(w_{parent(i)}(M)) + \sum_{j \in children(i)} out(w_j(M))$$

To elaborate, the message sending stage consists of erasing messages from the multiset in each inner membrane with label i, adding to each such multiset the objects a corresponding to messages (a, in_i) in the parent membrane (inner membrane with label $parent(i)$) and furthermore, adding the objects a corresponding to messages (a, out) in the children membranes (all inner membranes with label j, $j \in children(i)$).

Proposition 4 *If M is a configuration of Π then*

$$M \stackrel{\mathcal{R}}{\rightarrow}_{mpr} \rightarrow_{msg} N \text{ implies } \widetilde{N} \stackrel{\widetilde{\mathcal{R}}}{\rightarrow}_{\widetilde{mpr}} \rightarrow_{msg} \widetilde{M}.$$

If $\widetilde{N}$ is a configuration of $\widetilde{\Pi}$ then

$$\widetilde{N} \stackrel{\widetilde{\mathcal{R}}}{\rightarrow}_{\widetilde{mpr}} \rightarrow_{msg} \widetilde{M} \text{ implies } M \stackrel{\mathcal{R}}{\rightarrow}_{mpr} \rightarrow_{msg} N.$$

Proof We begin by describing some new notations. Consider a system of multisets of rules $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_m)$ for a P system Π with evolution rules $R_1^{\Pi}, \ldots, R_m^{\Pi}$. We define the following multisets of objects:

$$lhs^{obj}(\mathcal{R}_i), rhs^{obj}(\mathcal{R}_i), lhs^{out}(\mathcal{R}_i), rhs^{out}(\mathcal{R}_i), lhs^{in_j}(\mathcal{R}_i), rhs^{in_j}(\mathcal{R}_i)$$

such that, for u, v multisets of objects:

$$lhs^{obj}(\mathcal{R}_i)(a) = \sum_{r=u\to v\in R_i^{\Pi}} R_i(r)\cdot u(a); \qquad rhs^{obj}(\mathcal{R}_i)(a) = \sum_{r=u\to v\in R_i^{\Pi}} R_i(r)\cdot v(a)$$

$$lhs^{out}(\mathcal{R}_i)(a) = \sum_{r=u\to (v,out)\in R_i^{\Pi}} R_i(r)\cdot u(a); \quad rhs^{out}(\mathcal{R}_i)(a) = \sum_{r=u\to (v,out)\in R_i^{\Pi}} R_i(r)\cdot v(a)$$

$$lhs^{in_j}(\mathcal{R}_i)(a) = \sum_{r=u\to (v,in_j)\in R_i^{\Pi}} R_i(r)\cdot u(a); \quad rhs^{in_j}(\mathcal{R}_i)(a) = \sum_{r=u\to (v,in_j)\in R_i^{\Pi}} R_i(r)\cdot v(a)$$

We have the following properties:

- $lhs^{obj}(\mathcal{R}_i) = rhs^{obj}(\widetilde{\mathcal{R}_i})$ and $rhs^{obj}(\mathcal{R}_i) = lhs^{obj}(\widetilde{\mathcal{R}_i})$;
- $lhs^{out}(\mathcal{R}_i) = rhs^{in_i}(\widetilde{\mathcal{R}}_{parent(i)})$ and $rhs^{out}(\mathcal{R}_i) = lhs^{in_i}(\widetilde{\mathcal{R}}_{parent(i)})$;
- if $j \in children(i)$ then $lhs^{in_j}(\mathcal{R}_i) = rhs^{out}(\widetilde{\mathcal{R}_j})$ and $rhs^{in_j}(\mathcal{R}_i) = lhs^{out}(\widetilde{\mathcal{R}_j})$;
- $lhs(\mathcal{R}_i) = lhs^{obj}(\mathcal{R}_i) + lhs^{out}(\mathcal{R}_i) + \sum_{j\in children(i)} lhs^{in_j}(\mathcal{R}_i)$.

Now we can prove the statements of this Proposition. We prove only the first one; the proof of the second one is similar. If $M \xrightarrow{\mathcal{R}}_{mpr\to msg} N$ then there exists an intermediate configuration P such that $M \xrightarrow{\mathcal{R}}_{mpr} P$ and $P \to_{msg} N$. Then $\mathcal{R}_i$ are maximally valid in $w_i(M)$ and $w_i(P) = w_i(M) - lhs(\mathcal{R}_i) + rhs(\mathcal{R}_i)$. Since $w_i(M)$ is a multiset of objects, it follows that $obj(w_i(P)) = w_i(M) - lhs(\mathcal{R}_i) + rhs^{obj}(\mathcal{R}_i)$. If $j \in children(i)$ we have $in_j(w_i(P)) = rhs^{in_j}(\mathcal{R}_i)$ and moreover, $out(w_i(P)) = rhs^{out}(\mathcal{R}_i)$. Since $P \to_{msg} N$ we have $w_i(N) = obj(w_i(P)) + in_i(w_{parent(i)}(P)) + \sum_{j\in children(i)} out(w_j(P))$. Replacing $w_i(P)$, $w_{parent(i)}(P)$ and $w_j(P)$ we obtain

$$w_i(N) = w_i(M) - lhs(\mathcal{R}_i) + rhs^{obj}(\mathcal{R}_i) + rhs^{in_i}(\mathcal{R}_{parent(i)}) + \sum_{j\in children(i)} rhs^{out}(\mathcal{R}_j)$$

which is equivalent to

$$w_i(\widetilde{N}) = w_i(M) - lhs(\mathcal{R}_i) + lhs^{obj}(\widetilde{\mathcal{R}_i}) + lhs^{out}(\widetilde{\mathcal{R}_i}) + \sum_{j\in children(i)} lhs^{in_j}(\widetilde{\mathcal{R}_i})$$

i.e. $w_i(\widetilde{N}) = w_i(M) - lhs(\mathcal{R}_i) + lhs(\widetilde{\mathcal{R}_i})$. Therefore $\widetilde{\mathcal{R}_i}$ is valid in $w_i(\widetilde{N})$, $\forall i \in \{1,\ldots,m\}$. Suppose that $\widetilde{\mathcal{R}}$ is not reversely valid in $\widetilde{N}$. Then we have three possibilities, given by Definition 7. First, if there is $i \in \{1,\ldots,m\}$ and $\widetilde{r} = u \to v \in R_i^{\widetilde{\Pi}}$ such that $v \le w_i(\widetilde{N}) - lhs(\widetilde{\mathcal{R}_i})$ it means that $lhs(r) \le w_i(M) - lhs(\mathcal{R}_i)$, which contradicts the maximal validity of $\mathcal{R}_i$. Second, if there is $i \in \{1,\ldots,m\}$ and $\widetilde{r} = u \to (v, in_i) \in R_{parent(i)}^{\widetilde{\Pi}}$ such that $v \le w_i(\widetilde{N}) - lhs(\widetilde{\mathcal{R}_i})$ then again $lhs(r) \le w_i(M) - lhs(\mathcal{R}_i)$ (contradiction). The third situation leads to the same contradiction. Thus, there exists

an intermediate configuration Q in $\widetilde{\Pi}$ such that $\widetilde{N} \stackrel{\widetilde{\mathcal{R}}}{\rightarrow}_{\widetilde{mpr}} Q$. We have to show that $Q \rightarrow_{msg} \widetilde{M}$, i.e. to prove

$$w_i(\widetilde{M}) = obj(w_i(Q)) + in_i(w_{parent(i)}(Q)) + \sum_{j \in children(i)} out(w_j(Q))$$

Since $w_i(Q) = w_i(\widetilde{N}) - lhs(\widetilde{\mathcal{R}_i}) + rhs(\widetilde{\mathcal{R}_i})$ it follows that $obj(w_i(Q)) = w_i(M) - lhs(\mathcal{R}_i) + rhs^{obj}(\widetilde{\mathcal{R}_i})$. We also have that $in_i(w_{parent(i)}(Q)) = rhs^{in_i}(\widetilde{\mathcal{R}}_{parent(i)})$ and $out(w_j(Q)) = rhs^{out}(\widetilde{\mathcal{R}_j})$. So the relation we need to prove is equivalent to

$$w_i(\widetilde{M}) = w_i(M) - lhs(\mathcal{R}_i) + rhs^{obj}(\widetilde{\mathcal{R}_i}) + rhs^{in_i}(\widetilde{\mathcal{R}}_{parent(i)}) + \sum_{j \in children(i)} rhs^{out}(\widetilde{\mathcal{R}_j})$$

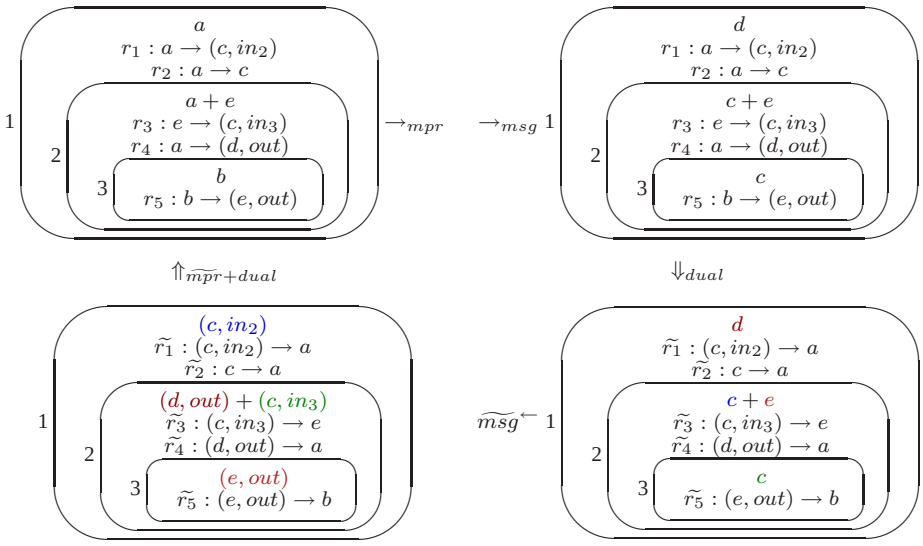
which is true because $lhs(\mathcal{R}_i) = lhs^{obj}(\mathcal{R}_i) + lhs^{out}(\mathcal{R}_i) + \sum_{j \in children(i)} lhs^{in_j}(\mathcal{R}_i)$.
□ □

5 An Alternative Approach

Another way to reverse a computation $N \stackrel{\mathcal{R}}{\rightarrow}_{mpr} \rightarrow_{msg} M$ is to move objects instead of moving rules. We start by reversing all rules of the P system Π; since these rules can be communication rules, by their reversal we do not obtain another P system. For example, a rule $a \rightarrow (b, out)$ yields $(b, out) \rightarrow a$, whose left-hand side contains the message out and therefore is not a rule. However, we can consider a notion of extended P system in which we allow rules to also have messages in their left-hand side. We move objects present in the membranes and transform them from objects to messages according to the rules of the membrane system. The aim is to achieve a result of form

$$M \stackrel{\mathcal{R}}{\rightarrow}_{mpr} N \rightarrow_{msg} P \text{ if and only if } \widetilde{P} \rightarrow_{\widetilde{msg}} \widetilde{N} \stackrel{\widetilde{\mathcal{R}}}{\rightarrow}_{\widetilde{mpr}} \widetilde{M}$$

An example illustrating the movement of the objects is the following:



where the "dual" movement $\rightarrow_{\widetilde{msg}}$ of objects between membranes is:

- d in membrane 1 $\xrightarrow{\text{called by rule } \tilde{r_4}}$ (d, out) in membrane 2;
- c in membrane 2 $\xrightarrow{\text{called by rule } \tilde{r_1}}$ (c, in_2) in membrane 1;
- e in membrane 2 $\xrightarrow{\text{called by rule } \tilde{r_5}}$ (e, out) in membrane 3;
- c in membrane 3 $\xrightarrow{\text{called by rule } \tilde{r_3}}$ (c, in_3) in membrane 2.

By applying the dual rules, messages are consumed and turned into objects, thus performing a reversed computation to the initial membrane.

6 Conclusion

In this paper, we solve the problem of finding all the configurations N of a P system which evolve to a given configuration M in a single step by introducing dual P systems. The case of P systems without communication rules is used as a stepping stone towards the case of P systems with simple communication rules. In the latter case, two approaches are presented: one where the rules are reversed and moved between membranes, and the other where the rules are only reversed. On dual membranes we employ a semantics which is surprisingly close to the one giving the maximally parallel rewriting (and message sending, if any).

The dual P systems open new research opportunities. A problem directly related to the subject of this paper is the predecessor existence problem in dynamical systems [1].

Dual P systems provide a simple answer, namely that a predecessor for a configuration exists if and only if there exists a system of multisets of rules which is reversely valid.

Dualising a P system is closely related to reversible computation [3]. Reversible computing systems are those in which every configuration is obtained from at most one previous configuration (predecessor). A paper which concerns itself with reversible computation in energy-based P systems is [2].

Further development will include defining dual P systems for P systems with general communication rules. Other classes of P systems will also be studied.

Acknowledgements. We thank the anonymous reviewers for their helpful comments. This research is partially supported by the grants CNCSIS Idei 402/2007 and CEEX 47/2005.

Bibliography

[1] C. Barretta, H.B. Hunt III, M.V. Marathea, S.S. Ravic, D.J. Rosenkrantz, R.E. Stearns, M. Thakurd. Predecessor Existence Problems for Finite Discrete Dynamical Systems. *Theoretical Computer Science* vol.386, 337, 2007.

[2] A. Leporati, C. Zandron, G. Mauri. Reversible P Systems to Simulate Fredkin Circuits. *Fundamenta Informaticae* vol.74, 529-548, 2006.

[3] K. Morita. Reversible Computing and Cellular Automata – A Survey. *Theoretical Computer Science* vol.395, 101 - 131, 2008.

[4] Gh. Păun. *Membrane Computing. An Introduction*. Springer, 2002.

Computing Solutions of #P-complete Problems by P Systems with Active Membranes

Artiom Alhazov[1], Liudmila Burtseva[1], Svetlana Cojocaru[1], Yurii Rogozhin[1,2]

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

[2]Rovira i Virgili University, Research Group on Mathematical Linguistics
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain

Membrane computing is a formal framework of distributed parallel multiset processing. Due to massive parallelism and exponential space some intractable computational problems can be solved by P systems with active membranes in polynomial number of steps. In this paper we generalize this approach from decisional problems to the computational ones, by providing a solution of a **#P**-Complete problem, namely to compute the permanent of a binary matrix.

1 Introduction

Membrane systems are a convenient framework of describing polynomial-time solutions to certain intractable problems in a massively parallel way. Division of membranes makes it possible to create exponential space in linear time, suitable for attacking problems in **NP** and even in **PSPACE**. Their solutions by so-called P systems with active membranes have been investigated in a number of papers since 2001, later focusing on solutions by restricted systems.

The description of rules in P systems with active membranes involves membranes and objects; the typical types of rules are (a) object evolution, $(b), (c)$ object communication, (d) membrane dissolution, $(e), (f)$ membrane division, see Subsection 2.2. Since membrane systems are an abstraction of living cells, the membranes are arranged hierarchically, yielding a tree structure. A membrane is called elementary if it is a leaf of this tree, i.e., if it does not contain other membranes.

The first efficient *semi–uniform solution* to SAT was given by Gh. Păun in [4], using division for non–elementary membranes and three electrical charges. This result was

improved by Gh. Păun, Y. Suzuki, H. Tanaka, and T. Yokomori in [5] using only division for elementary membranes.

Different efficient *uniform solutions* have been obtained in the framework of recognizer P systems with active membranes, with polarizations and only using division rules for elementary membranes ([7], [6], [3], [9], [1], [8], and [11]).

The goal of this paper is to generalize the approach from decisional problems to the computational ones, by considering a **#P**-Complete (pronounced sharp-P complete) problem of computing the *permanent* of a binary matrix; see also section 1.3.7 in [13] for a presentation of Complexity Theory of counting problems.

Let us cite [14] for additional motivation:

While 3SAT and the other problems in **NP**-Complete are widely assumed to require an effort at least proportional to 2^n, where n is a measure of the size of the input, the problems in **#P**-Complete are harder, being widely assumed to require an effort proportional to $n2^n$.

While attacking **NP** complexity class by P systems with active membranes have been often motivated by $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ problem, we recall from [15] the following fact:

If the permanent can be computed in polynomial time by any method, then **FP=#P** which is an even stronger statement than **P= NP**.

Here, by "any method" one understands "... on sequential computers" and **FP** is the set of polynomial-computable functions.

2 Definitions

Membrane computing is a recent domain of natural computing started by Gh. Păun in 1998. The components of a membrane system are a cell-like membrane structure, in the regions of which one places multisets of objects which evolve in a synchronous maximally parallel manner according to given evolution rules associated with the membranes.

2.1 Computing by P systems Let O be a finite set of elements called objects. In this paper, like it is standard in membrane systems literature, a multiset of objects is denoted by a string, so the multiplicity of object is represented by number of its occurrences in the string. The empty multiset is thus denoted by the empty string, λ.

To speak about the result of the computation of a P system we need the definition of a P system with output.

Definition 1 *A P system with output,* Π*, is a tuple*

$\Pi = \big(O, T, H, E, \mu, w_1, \cdots, w_p, R, i_0\big)$, *where:*

- O *is the working alphabet of the system whose elements are called objects.*
- $T \subseteq O$ *is the output alphabet.*
- H *is an alphabet whose elements are called labels.*
- E *is the set of polarizations.*
- μ *is a membrane structure (a rooted tree) consisting of* p *membranes injectively labelled by elements of* H.
- w_i *is a string representing an* initial *multiset over* O *associated with membrane* i, $1 \leq i \leq p$.
- R *is a finite set of rules defining the behavior of objects from* O *and membranes labelled by elements of* H.
- i_0 *identifies the output region.*

A configuration of a P system is its "snapshot", i.e., the current membrane structure and the multisets of objects present in regions of the system. While initial configuration is $C_0 = (\mu, w_1, \cdots, w_p)$, each subsequent configuration C' is obtained from the previous configuration C by maximally parallel application of rules to objects and membranes, denoted by $C \Rightarrow C'$ (no further rules are applicable together with the rules that transform C into C'). A computation is thus a sequence of configurations starting from C_0, respecting relation $\Rightarrow$ and ending in a halting configuration (i.e., such one that no rules are applicable).

The P systems of interest here are those for which all computations give the same result. This is because it is enough to consider one computation to obtain all information about the result.

Definition 2 *A P system with output is* confluent *if (a) all computations halt; and (b) at the end of all computations of the system, region* i_0 *contains the same multiset of objects from* T.

In this case one can say that the multiset mentioned in (b) is the result given by a P system, so this property is already sufficient for convenient usage of P systems for computation.

However, one can still speak about a stronger property: a P system is *strongly confluent* if not only the result of all computation is the same, but also the halting configuration is the same. A yet stronger property is determinism: a P system is called *deterministic* if it only has one computation.

In what follows we will represent computational problems by triples: domain, range and the function (from that domain into that range) that needs to be computed. The

notation $\mathbf{PMC}^*_{\mathcal{R}}$ of the class of problems that are polynomially computable by semi-uniform families of P systems with active membranes has been introduced by M.J. Pérez Jiménez and his group, see, e.g., [10]. The definition below generalizes it from decisional problems to the computational ones.

Definition 3 *Let $X = (I_X, F, \theta_X)$ be a computational problem: $\theta_X : I_X \to F$. We say that X is solvable in polynomial time by a (countable) family $\mathcal{R}$ of confluent P systems with output $\mathbf{\Pi} = (\Pi(u))_{u \in I_X}$, and we denote this by $X \in \mathbf{PMC}^*_{\mathcal{R}}$, if the following are true.*

1. *The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines, i.e., there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(u)$ from the instance $u \in I_X$.*
2. *The family $\mathbf{\Pi}$ is polynomially bounded: for some polynomial function $p(n)$ for each instance $u \in I_X$ of the problem, all computations of $\Pi(u)$ halt in, at most, $p(|u|)$ steps.*
3. *There exists a polynomial-time computable function dec such that the family $\mathbf{\Pi}$ correctly answers X with respect to (X, dec): for each instance of the problem $u \in I_X$, the function dec applied to the result given by $\Pi(u)$ returns exactly $\theta_X(u)$.*

We say that the family $\mathbf{\Pi}$ is a *semi–uniform solution* of the problem X.

Now we additionally consider input into P systems and we deal with P systems solving computational problems in a *uniform* way in the following sense: all instances of the problem with the same *size* (according to a previously fixed polynomial time computable criterion) are processed by the same system, on which an appropriate input, representing the specific instance, is supplied.

If w is a multiset over the input alphabet $\Sigma \subseteq O$, then the *initial configuration* of a P system Π with an input w over alphabet Σ and input region i_Π is

$$(\mu, w_1, \cdots, w_{i_\Pi - 1}, w_{i_\Pi} \cup w, w_{i_\Pi + 1} \cdots, w_p).$$

In the definition below we present the notation $\mathbf{PMC}_{\mathcal{R}}$ of the class of problems that are polynomially computable by uniform families of P systems with active membranes introduced by M.J. Pérez Jiménez and his group, see, e.g., [10], generalized from decisional problems to the computational ones.

Definition 4 *Let $X = (I_X, F, \theta_X)$ be a computational problem. We say that X is solvable in polynomial time by a family $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbb{N}}$ of confluent membrane systems with input, and we denote it by $X \in \mathbf{PMC}_{\mathcal{R}}$, if*

1 *The family* $\mathbf{\Pi}$ *is polynomially uniform by TM: some deterministic TM constructs in polynomial time the system* $\Pi(n)$ *from* $n \in \mathbb{N}$.

2 *There exists a pair* (cod, s) *of polynomial-time computable functions whose domain is* I_X *and a polynomial-time computable function* dec *whose range is* F*, such that for each* $u \in I_X$*,* $s(u)$ *is a natural number,* $cod(u)$ *is an input multiset of the system* $\Pi(s(u))$*, verifying the following:*

2a *The family* $\mathbf{\Pi}$ *is polynomially bounded with respect to* (X, cod, s)*; that is, there exists a polynomial function* $p(n)$ *such that for each* $u \in I_X$ *every computation of the system* $\Pi(s(u))$ *with input* $cod(u)$ *halts in at most* $p(|u|)$ *steps.*

2b *There exists a polynomial-time computable function* dec *such that the family* $\mathbf{\Pi}$ *correctly answers* X *with respect to* (X, cod, s, dec)*: for each instance of the problem* $u \in I_X$*, the function* dec*, being applied to the result given by* $\Pi(s(u))$ *with input* $cod(u)$*, returns exactly* $\theta_X(u)$.

We say that the family $\mathbf{\Pi}$ is a *uniform solution* to the problem X.

2.2 P systems with active membranes To speak about P systems with active membranes, we need to specify the rules, i.e., the elements of the set R in the description of a P system. They can be of the following forms:

(a) $[\, a \to v \,]_h^e$,
for $h \in H, e \in E, a \in O, v \in O^*$
(object evolution rules, associated with membranes and depending on the label and the polarization of the membranes, but not directly involving the membranes, in the sense that the membranes are neither taking part in the application of these rules nor are they modified by them);

(b) $a[\ \ ]_h^{e_1} \to [\, b \,]_h^{e_2}$,
for $h \in H, e_1, e_2 \in E, a, b \in O$
(communication rules; an object is introduced into the membrane; the object can be modified during this process, as well as the polarization of the membrane can be modified, but not its label);

(c) $[\, a \,]_h^{e_1} \to [\ \ ]_h^{e_2} b$,
for $h \in H, e_1, e_2 \in E, a, b \in O$
(communication rules; an object is sent out of the membrane; the object can be modified during this process; also the polarization of the membrane can be modified, but not its label);

(d) $[\, a \,]_h^e \to b$,
for $h \in H, e \in E, a, b \in O$
(dissolving rules; in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified);

(e) $[\, a \,]_h^{e_1} \to [\, b \,]_h^{e_2} [\, c \,]_h^{e_3}$,
for $h \in H, e_1, e_2, e_3 \in E, a, b, c \in O$

(division rules for elementary membranes; in reaction with an object, the membrane is divided into two membranes with the same label, possibly of different polarizations; the object specified in the rule is replaced in the two new membranes by possibly new objects).

(f_0) $[\,[\,]_{h_1}[\,]_{h_2}\,]_{h_0} \rightarrow [\,[\,]_{h_1}\,]_{h_0}\,[\,[\,]_{h_2}\,]_{h_0}$,
for $h_0, h_1, h_2 \in H$
(polarizationless division rules for non–elementary membranes. If the membrane with label h_0 contains other membranes than those with labels h_1, h_2, these membranes and their contents are duplicated and placed in both new copies of the membrane h_0; all membranes and objects placed inside membranes h_1, h_2, as well as the objects from membrane h_0 placed outside membranes h_1 and h_2, are reproduced in the new copies of membrane h_0).

In this paper we do not need non–elementary membrane division, dissolution or rules that bring an object inside a membrane; they are mentioned in the definition for completeness.

The rules of type (a) are considered to only involve objects, while all other rules are assumed to involve objects and membranes mentioned in their left-hand side. An application of a rule consists in subtracting a multiset described in the left-hand side from a corresponding region (i.e., associated to a membrane with label h and polarization e for rules of types (a) and (d), or associated to a membrane with label h and polarization e_1 for rules of type (c) and (e), or immediately outer of such a membrane for rules of type (b)), adding a multiset described in the right-hand side of the rule to the corresponding region (that can be the same as the region from where the left-hand side multiset was subtracted, immediately inner or immediately outer, depending on the rule type), and updating the membrane structure accordingly if needed (changing membrane polarization, dividing or dissolving a membrane).

The rules can only be applied simultaneously if they involve different objects and membranes (we repeat that rules of type (a) are not considered to involve a membrane), and such parallelism is maximal if no further rules are applicable to objects and membranes that were not involved.

2.3 Permanent of a matrix The complexity class #P, see [16], was first defined in [12] in a paper on the computation of the permanent.

Definition 5 *Let S_n be the set of permutations of integers from 1 to n, i.e., the set of bijective functions $\sigma : \{1, \cdots, n\} \rightarrow \{1, \cdots, n\}$. The permanent of a matrix $A = (a_{i,j})_{1 \leq i,j \leq n}$ is defined as*

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}.$$

Informally, consider a combination of n matrix elements containing one element from every row and one element from every column. The permanent is the sum over all such combinations of the product of the combination's elements.

A matrix is binary if its elements are either 0 or 1. In this case, the permanent is the number of combinations of n matrix elements with value 1, containing one element from each row and one element from each column. For example,

$$perm \begin{pmatrix} 1\ 0\ 1 \\ 0\ 1\ 0 \\ 1\ 0\ 1 \end{pmatrix} = 2.$$

Unlike the determinant of a matrix, the permanent cannot be computed by Gauss elimination.

3 Results

Theorem 1 *The problem of computing a permanent of a binary matrix is solvable in polynomial time by a uniform family of deterministic P systems with active membranes with two polarizations and rules of types* $(a), (c), (e)$.

Proof Let $A = (a_{i,j})$ be an $n \times n$ matrix. We define $N = \lceil \log_2(n) \rceil$, and $n' = 2^N < 2n$ is the least power of two not smaller then n. The input alphabet is $\Sigma(n) = \{\langle i, j\rangle \mid 1 \leq i \leq n,\ 1 \leq j \leq n\}$, and the matrix A is given as a multiset $w(A)$ containing for every element $a_{i,j} = 1$ of the matrix one symbol $\langle i, j\rangle$. Let the output alphabet be $T = \{o\}$, we will present a P system $\Pi(n)$ giving $o^{perm(A)}$ as the result when given input $w(A)$ in region $i_{\Pi(n)} = 2$.

$$\begin{aligned}
\Pi(n) &= \big(O, T, H, E, \mu, w_1, w_2, R, 1\big), \\
O &= \Sigma(n) \cup T \cup \{c\} \cup \{d_i, a_i \mid 0 \leq i \leq Nn\} \cup \{D_i \mid 0 \leq i \leq n+1\} \\
&\quad \cup \{\langle i, j, k, l\rangle \mid 0 \leq i \leq Nn-1,\ 0 \leq j \leq n-1,\ 0 \leq k \leq Nn-1, \\
&\qquad 0 \leq l \leq n'-1\}, \\
\mu &= [\,[\,]_2^0\,]_1^0,\ H = \{1, 2\},\ E = \{0, 1\}, \\
w_1 &= \lambda,\ w_2 = d_0.
\end{aligned}$$

and the rules are presented and explained below.

A1 $[\,\langle i, j\rangle \rightarrow \langle Ni - 1, j - 1, Nn - 1, 0\rangle\,]_2^0, 1 \leq i \leq n, 1 \leq j \leq n$

Preparation of the input objects: tuple representation. Informal meaning of the tuple components is 1) number of steps remaining until row i is processed, 2) column number,

starting from 0, 3) number of steps remaining until all rows are processed, 4) will be used for memorizing the chosen column.

A2 $[\ d_i\ ]_2^e \rightarrow [\ d_{i+1}\ ]_2^0 [\ d_{i+1}\ ]_2^1, 0 \leq i \leq Nn-1, e \in E$

Division of the elementary membrane for Nn times.

A3 $[\ \langle i,j,k,l\rangle \rightarrow \langle i-1, j, k-1, 2l+e\rangle\ ]_2^e$,
$0 \leq i \leq Nn-1$, i is not divisible by N,
$0 \leq j \leq n-1, 1 \leq k \leq Nn-1, 0 \leq l \leq (n-1-e)/2, e \in E$

For i times, during $N-1$ steps input objects corresponding to row i memorize the polarization history. The binary representation of the chosen column for the current row corresponds to the history of membrane polarizations during N steps.

A4 $[\ \langle i,j,k,l\rangle \rightarrow \lambda\ ]_2^e$,
$0 \leq i \leq Nn-1, 0 \leq j \leq n-1, 1 \leq k \leq Nn-1$,
$(n-1-e)/2 \leq l \leq n'/2-1, e \in E$

Erase all input objects if the chosen column is invalid, i.e., its number exceeds $n-1$.

A5 $[\ \langle i,j,k,l\rangle \rightarrow \langle i-1, j, k-1, 0\rangle\ ]_2^e$,
$1 \leq i \leq Nn-1, 0 \leq j \leq n-1, j \neq 2l+e$,
$0 \leq k \leq Nn-1, 0 \leq l \leq (n-1-e)/2, e \in E$

If element's row is not reached and element's column is not chosen, proceed to the next row.

A6 $[\ \langle i,j,k,l\rangle \rightarrow \lambda\ ]_2^e$,
$1 \leq i \leq Nn-1, 0 \leq j \leq n-1, j = 2l+e$,
$0 \leq k \leq Nn-1, 0 \leq l \leq (n-1-e)/2, e \in E$

Erase the chosen column, except the chosen element.

A7 $[\ \langle 0,j,k,l\rangle \rightarrow \lambda\ ]_2^e$,
$0 \leq j \leq n-1, j \neq 2l+e$,
$0 \leq k \leq Nn-1, 0 \leq l \leq (n-1-e)/2, e \in E$

Erase the chosen row, except the chosen element.

A8 $[\ \langle 0, j, k, l\rangle \to a_{k-1}\ ]_2^e$,
$0 \le j \le n-1$, $j = 2l + e$,
$0 \le k \le Nn - 1$, $0 \le l \le (n - 1 - e)/2$, $e \in E$

If chosen element is present (i.e., it has value 1 and its column has not been chosen before), produce object a_{k-1}.

A9 $[\ a_k \to a_{k-1}\ ]_2^e$, $1 \le k \le Nn - 1$, $e \in E$

Objects a_k wait until all rows are processed. Then a membrane represents a solution if n copies of a_0 are present.

B1 $[\ d_{Nn} \to D_{1-e}c^{n+e}\ ]_2^e$, $e \in E$

If polarization is 0, produce n copies of object c and a counter D_1. Otherwise, produce one extra copy of c and set the counter to D_0; this will reduce to the previous case in one extra step.

B2 $[\ c\ ]_2^1 \to [\ \ ]_2^0 c$

B3 $[\ a_0\ ]_2^0 \to [\ \ ]_2^1 a_0$

B4 $[\ D_i \to D_{i+1}\ ]_2^1$, $0 \le i \le n$

Each object a_0 changes polarization to 1, the counter D_i counts this, and then object c resets the polarization to 0.

B5 $[\ D_{n+1}\ ]_2^1 \to [\ \ ]_2^0 o$

If there are n chosen elements with value 1, send one object o out.

The system is deterministic. Indeed, for any polarization and any object (other than d_i, $i < Nn$, c, a_0 or D_{n+1}), there exist at most one rule of type (a) and no other associated rules. As for the objects in parentheses above, they have no rules of type (a) associated with them and they cause a well-observed deterministic behavior of the system: division rules are applied during the first Nn steps; then, depending on the polarization, symbols a_0 or c are sent out; finally, wherever D_{n+1} is produced, it is sent out.

The system computes the permanent of a matrix in at most $n(2 + N) + 1 = O(n \log n)$ steps. Indeed, first Nn steps correspond to membrane divisions corresponding to finding all permutations of S_n, see Definition 5, while the following steps correspond to

counting the number of non-zero entries of the matrix associated to these permutations (there are at most $2n + 1$ of them since the system counts to at most n and each count takes two steps; one extra step may be needed for a techinical reasons: to reset to 0 the polarization of membranes that had polarization 1 after the first Nn steps). □

It should be noted that requirement that the output region is the environment (typically done for decisional problem solutions) has been dropped. This makes it possible to give non-polynomial answers to the permanent problem (which is a number between 0 and $n!$) in polynomial number of steps without having to recall from [2] rules sending objects out that work in parallel.

4 Discussion

In this paper we presented a solution to the problem of computing a permanent of a binary matrix by P systems with active membranes, namely with two polarizations and rules of object evolution, sending objects out and membrane division. This problem is known to be **#P**-Complete. The solution has been preceded by the framework that generalizes decisional problems to computing functions: now the answer is much more than one bit. This result suggests that P systems with active membranes without non–elementary membrane division still compute more than decisions of the problems in **NP**.

Acknowledgments. All authors gratefully acknowledge the support by the Science and Technology Center in Ukraine, project 4032. Yurii Rogozhin gratefully acknowledges the support of the European Commission, project MolCIP, MIF1-CT-2006-021666.

Bibliography

[1] A. Alhazov, C. Martín–Vide, L. Pan, Solving Graph Problems by P Systems with Restricted Elementary Active Membranes. *Aspects of Molecular Computing* (N. Jonoska, Gh. Păun, G. Rozenberg, Eds.), Lecture Notes in Computer Science **2950**, 2004, 1–22.

[2] A. Alhazov, L. Pan, Gh. Păun, Trading Polarizations for Labels in P Systems with Active Membranes, *Acta Informaticae* **41**, 2-3, 2004, 111-144.

[3] M. A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, A Fast P System for Finding a Balanced 2-partition, *Soft Computing* **9**, 9, 2005, 673–678.

[4] Gh. Păun, P Systems with Active Membranes: Attacking **NP**–complete Problems, *Journal of Automata, Languages and Combinatorics*, **6**, 1, 2001, 75–90.

[5] Gh. Păun, Y. Suzuki, H. Tanaka, T. Yokomori, On the Power of Membrane Division in P Systems, *Theoretical Computer Science* **324**, 1, 2004, 61–85.

[6] M.J. Pérez-Jiménez, A. Riscos-Núñez, Solving the Subset-Sum Problem by Active Membranes, *New Generation Computing* **23**, 4, 2005, 367–384.

[7] Pérez-Jiménez, M.J., Riscos-Núñez, A., A Linear–time Solution to the Knapsack Problem Using P Systems with Active Membranes, *Membrane Computing, International Workshop, WMC 2003* (C. Martín-Vide, Gh. Păun, G. Rozenberg, A. Salomaa, Eds.), Lecture Notes in Computer Science **2933** (2004), 250–268.

[8] M.J. Pérez-Jiménez, F.J., Romero-Campero, Solving the Bin Packing Problem by Recognizer P Systems with Active Membranes, *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini, Eds.), (RGNC Report **01/04**, University of Seville, 2004, 414–430.

[9] M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, Complexity Classes in Cellular Computing with Membranes, *Natural Computing* **2**, 3, 2003, 265-285.

[10] M.J. Pérez-Jimé'nez, A. Romero-Jiménez, F. Sancho-Caparrini, Solving VALIDITY Problem by Active Membranes with Input, *Brainstorming Week on Membrane Computing*, Tarragona, 2003 (M. Cavaliere, C. Martín-Vide, Gh. Păun, Eds.), Technical Report 26, Rovira i Virgili University, Tarragona, 2003, 279–290.

[11] M.J. Pérez-Jiménez, F.J. Romero–Campero, Attacking the Common Algorithmic Problem by Recognizer P Systems. *Machines, Computations and Universality*, (M. Margenstern, Ed.) Lecture Notes in Computer Science **3354**, 2005, 304-315.

[12] L. G. Valiant, The Complexity of Computing the Permanent, *Theoretical Computer Science* **8**, Elsevier, 1979, 189–201.

[13] I. Wegener, *Complexity Theory: Exploring the Limits of Efficient Algorithms*, Springer-Verlag, 2005.

[14] R. M. Williams, D. H. Wood, Exascale Computer Algebra Problems Interconnect with Molecular Reactions and Complexity Theory, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **44**, 1999, 267–275.

[15] `http://en.wikipedia.org/wiki/Permanent` (updated 05.05.2008)

[16] `http://en.wikipedia.org/wiki/Sharp-P` (updated 13.12.2007)

Fast synchronization in P systems

Artiom Alhazov[1], Maurice Margenstern[2], Sergey Verlan[1,3]

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

[2]Université Paul Verlaine - Metz, LITA, EA 3097, IUT de Metz
Ile du Saulcy, 57045 Metz Cédex, France
margens@univ-metz.fr

[3]Université Paris 12, LACL, Département Informatique,
61 av. Général de Gaulle, 94010 Créteil, France
verlan@univ-paris12.fr

We consider the problem of synchronizing the activity of all membranes of a P system. After pointing at the connection with a similar problem dealt with in the field of cellular automata where the problem is called the *firing squad synchronization problem*, $FSSP$ for short, we provide two algorithms to solve this problem. One algorithm is non-deterministic and works in $2h+3$, the other is deterministic and works in $3h+3$, where h is the height of the tree describing the membrane structure.

1 Introduction

The synchronization problem can be formulated in general terms with a wide scope of application. We consider a system constituted of explicitly identified elements and we require that starting from an initial configuration where one element is distinguished, after a finite time, all the elements which constitute the system reach a common feature, which we call **state**, all at the same time and the state was never reached before by any element.

This problem is well known for cellular automata, where it was intensively studied under the name of the *firing squad synchronization problem* (FSSP): a line of soldiers have to fire at the same time after the appropriate order of a general which stands at one end of the line, see [2, 5, 4, 9–11]. The first solution of the problem was found by Goto, see [2]. It works on any cellular automaton on the line with n cells in the minimal time, $2n-2$ steps, and requiring several thousands of states. A bit later, Minsky found his famous solution which works in $3n$, see [5] with a much smaller number of states, 13 states. Then, a race to find a cellular automaton with the smallest number of states

which synchronizes in $3n$ started. See the above papers for references and for the best results and for generalizations to the planar case, see [9] for results and references.

The synchronization problem appears in many different contexts, in particular in biology. As P systems model the work of a living cell constituted of many micro-organisms, represented by its membranes, it is a natural question to raise the same issue in this context. Take as an example the meiosis phenomenon, it probably starts with a synchronizing process which initiates the division process. Many studies have been dedicated to general synchronization principles occurring during the cell cycle; although some results are still controversial, it is widely recognised that these aspects might lead to an understanding of general biological principles used to study the normal cell cycle, see [8].

We may translate FSSP in P systems terms as follows. Starting from the initial configuration where all membranes, except the root, contain same objects the system must reach a configuration where all membranes contain a distinguished symbol, F. Moreover, this symbol must appear in all membranes only during at the synchronization time.

The synchronization problem as defined above was studied in [1] for two classes of P systems: transitional P systems and P systems with priorities and polarizations. In the first case, a non-deterministic solution to FSSP was presented and for the second case a deterministic solution was found. These solutions need a time $3h$ and $4n + 2h$ respectively, where n is the number of membranes of a P system and h is the depth of the membrane tree.

In this article we significantly improve the previous results in the non-deterministic case. In the deterministic case, another type of P system was considered and this permitted to improve the parameters. The new algorithms synchronize the corresponding P systems in $2h + 3$ and $3h + 3$ respectively.

2 Definitions

In the following we briefly recall the basic notions concerning P systems. For more details we refer the reader to [6] and [12].

A transitional P system of degree n is a construct

$$\Pi = (O, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, i_0),$$

where:

1. O is a finite alphabet of symbols called objects,
2. μ is a membrane structure consisting of n membranes labelled in a one-to-one manner by $1, 2, \ldots, n$ (the outermost membrane is called the *skin* membrane),

3. $w_i \in O^*$, for each $1 \leq i \leq n$ is a multiset of objects associated with the region i (delimited by membrane i),
4. for each $1 \leq i \leq n$, R_i is a finite set of rules associated with the region i which have the following form $u \rightarrow v_1, tar_1; v_2, tar_2; \ldots; v_m, tar_m$, where $u \in O^+$, $v_i \in O$ and $tar_i \in \{in, out, here, in!\}$,
5. i_0 is the label of an elementary membrane of μ that identifies the output region.

A transitional P system is defined as a computational device consisting of a set of n hierarchically nested membranes that identify n distinct regions (the membrane structure μ) where, to each region i, a multiset of objects w_i and a finite set of evolution rules R_i, $1 \leq i \leq n$ are assigned.

An evolution rule $u \rightarrow v_1, tar_1; v_2, tar_2; \ldots; v_m, tar_m$ rewrites u by $v_1, \ldots, v_m$ and moves each v_j accordingly to the target tar_j. If the tar_j target is $here$, then v_j remains in membrane i. Target $here$ can be omitted in the notation. If the target tar_j is out, then v_j is sent to the parent membrane of i. If the target tar_j is in, then v_j is sent to any inner membrane of i chosen non-deterministically. If the target tar_j is equal to $in!$, then v_j is sent to all inner membranes of i (a necessary number of copies is made).

A computation of the system is obtained by applying the rules in a non-deterministic maximally parallel manner. Initially, each region i contains the corresponding finite multiset w_i.

A computation is successful if starting from the initial configuration it reaches a configuration where no rule can be applied. The result of a successful computation is the natural number obtained by counting the objects that are presented in region i_0. Given a P system Π, the set of natural numbers computed in this way by Π is denoted by $N(\Pi)$. In the sequel we shall omit i_0 since it is irrelevant for FSSP.

A *transitional P system with promoters and inhibitors* is a construct

$$\Pi = (O, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, i_0),$$

defined as in the previous definition, where the set of rules may contain rules of the form

$$u \rightarrow v_1, tar_1; v_2, tar_2; \ldots; v_m, tar_m \mid_{P, \neg Q},$$

where $P \in O$ is the *promoter*, $Q \in O$ is the *inhibitor*, $tar_i \in \{in, out, here, in!\}$, $u \in O^+$ and $v_i \in O$. If P and/or Q are absent, we shall omit them. The meaning of promoter and inhibitor (if present in a rule) is following: the rule is not applicable unless the promoter object exists in the current membrane, while the rule is applicable unless the inhibitor object is present in the current membrane.

As above, the target $here$, which can be omitted in the notation, means that the object remains in the current membrane and the $in!$ target sends the corresponding object to all inner membranes at the same time, making the right number of copies.

Each rule may be applied if and only if the corresponding membrane does contain the object P and does not contain the object Q. A computational step is obtained by applying the rules in a non-deterministic maximally parallel manner.

In the sequel, we will use transitional P systems without a distinguished compartment as an output, i_0, as this is not relevant for FSSP.

We translate the FSSP to P systems as follows:

Problem 1 *For a class of P systems $\mathcal{C}$ find two multisets $\mathcal{W}, \mathcal{W}' \in O^*$, and two sets of rules $\mathcal{R}$, $\mathcal{R}'$ such that for any P system $\Pi \in \mathcal{C}$ of degree $n \geq 2$ having*

$w_1 = \mathcal{W}'$, $R_1 = \mathcal{R}'$, $w_i = \mathcal{W}$ and $R_i = \mathcal{R}$ for all i in $\{2..n\}$, assuming that the skin membrane has the number 1

it holds

- *If the skin membrane is not taken into account, then the initial configuration of the system is stable (cannot evolve by itself).*
- *If the system halts, then all membranes contain the designated symbol F which appears only at the last step of the computation.*

3 Non-deterministic case

In this section we discuss a non-deterministic solution to the FSSP using transitional P systems. The main idea of such a synchronization is based on the fact that if a signal is sent from the root to a leaf then it will take at most $2h$ steps to reach a leaf and return back to the root. In the meanwhile, the root may guess the value of h and propagate it step by step down the tree. This takes also $2h$ steps: h to guess the root, and h to end the propagation and synchronize. Hence, if the signal sent to the leaf, having depth $d \leq h$, returns at the same moment that the root ended the propagation, then the root guessed the value d. Now, in order to finish the construction it is sufficient to cut off cases when $d < h$.

In order to implement the above algorithm in transitional P systems we use following steps.

- Mark leaves and nodes (nodes by $\bar{S}$ and leaves by S).
- From the root, send a copy of symbol a down. Any inner node must take one a in order to pass to state S'. If some node is not passed to state S' then when the signal c will come inside, it will be transformed to $\#$.
- Then end of the guess is marked by signal c. Symbols S in leaves are transformed to S''' and those in inner nodes to S''.

- In the meanwhile the height is computed with the help of C_3. If a smaller height $d \leq h$ is obtained at the root node, then either the symbol C_3 will arrive to the root node and it will contain some symbols b – then the symbol $\#$ will be introduced at the root node, or the guessed value will be d and then there will be an inner node with $\bar{S}$ or a leaf with S (because we have at most d letters a) which leads to the introduction of $\#$ in corresponding node.

Now let us present the system in details.

Let $\Pi = (O, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n)$ the P system to be synchronized, where i_0 is not mentioned as it is not relevant for the synchronization. To solve the synchronization problem, we make the following assumptions on the objects, the membranes and the rules. We consider that μ is an arbitrary membrane structure and

$$O = \{S, \bar{S}, S_1, S_2, S_3, C_1, C_2, C_3, S', S'', S''', a, b, c, F, \#\}.$$

We also assume that $w_1 = \{S_1\}$, where w_1 is the contents of the skin membrane and that all other membranes are empty. The sets of rules, $R_1, \ldots, R_n$ are all equal and they are described below.

Start:

$$S_1 \to S_2; C_2; S, in!; C_1, in \tag{1}$$

Propagation of S:

$$S \to \bar{S}; S, in! \tag{2}$$

Root counter (guess):

$$S_2 \to S_2; b; a, in! \qquad S_2 \to S_3; c, in! \tag{3}$$

Propagate a:

$$\bar{S}a \to S' \qquad a \to b; a, in! \tag{4}$$

Propagate c:

$$cS' \to S''; c, in! \qquad cSa \to S''' \tag{5}$$

Decrement:

$$S''b \to S'' \qquad S'''a \to S''' \tag{6}$$

$$S'' \to F \qquad S''' \to F \tag{7}$$

$$S_3 b \to S_3 \tag{8}$$

Height computing:

$$C_1 \to C_1, in \qquad C_2 \to C_2, in \tag{9}$$
$$C_1 C_2 \to C_3 \qquad C_2 \to \# \tag{10}$$
$$C_3 \to C_3, out \tag{11}$$

Root firing:

$$C_3 S_3 \to F \tag{12}$$

Traps:

$$c\bar{S} \to \# \qquad cS \to \# \qquad C_3 \to \# \tag{13}$$
$$aF \to \# \qquad bF \to \# \qquad \# \to \# \tag{14}$$

We affirm that the system Π has the desired behavior. Indeed, let us consider the functioning of this system.

Rule (1) produces objects S, C_1, C_2 and S_2. Object S will propagate down the tree structure by rule (2), leaving $\bar{S}$ in all intermediate nodes and S in the leaves. Objects C_1 and C_2 will be used to count the time corresponding to twice the depth d of some elementary membrane by rules (9)-(11) (trying to guess the maximal depth). Finally, object S_2 will produce objects b in some multiplicity by rules (3).

Together with objects b, objects a are produced by the first rule from (3), and they propagate down the tree structure by (4), one copy being subtracted at each level.

After the root finishes guessing the depth (second rule of (3)), object c propagates down the tree structure by (5), producing objects S'' at intermediate nodes and objects S''' at leaves; recall that the root has object S_3. These three objects perform the countdown (and then the corresponding nodes fire) by rules (6). As for the root, at firing by (12) it also checks that the timing matches twice the depth of the node visited by C_1 and C_2. The rules (13)-(14) handle possible cases of behavior of the system, not leading to the synchronization.

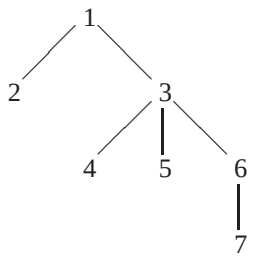
Now we shall present a more formal proof of the assertion above. We have the following claims.

- We claim that the symbol C_3 will appear at the root node at the time $2d+2$, $d \leq h$, where h is the height of the membrane structure and d is the depth of the leaf visited by C_1. Indeed, by rules (9) symbols C_1 and C_2, initially created by rule (1), go down until they reach a leaf. If they do not reach the same leaf, then the symbol $\#$ is introduced by C_2. The symbol C_2 reaches the leaf (of depth d) after $d+1$ steps. After that C_1 and C_2 are transformed to the symbol C_3 (1 step) which starts travelling up until it reaches the root node (d steps).

- We claim that all nodes inner nodes will be marked by $\bar{S}$ and the leaves will be marked by S. Indeed, rule (2) permits to implement this behavior.
- Let $d+2$, $0 \leq d \leq h$ be the moment when the root stops the guess of the tree height (the second rule from (3) has been applied). At this moment the contents of w_1 is $S_3 b^d$ and c starts to be propagated. Now consider any node x except the root. We claim that:
  If x is of depth i then symbol c will reach x at time $d+i+1$ and the number of letters a (respectively letters b) present at x if it is a leaf (respectively inner node) is $a^{d \ominus i}$ (respectively $b^{d \ominus (i+1)}$), where $\ominus$ denotes the positive substraction.
  The proof of this assertion may be done by induction. Initially, at step $d+2$, symbol c is present in all nodes of depth 1. Let x be such a node. If x is a leaf, then it received d copies of a. Otherwise, if x is an inner node, it must contain $d \ominus 1$ letters b (d letters a reached this node and all of them except one were transformed to b). The induction step is trivial since the letter c propagates each step down the tree and because the number of letters a reaching a depth i is smaller by one than the number of a reaching the depth $i-1$.
- From the above assertion it is clear that all nodes at time $2d+2$ will reach the configuration where there are no more letters b and a. Hence, all nodes, including the root node, up to depth d will synchronize at time $2d+3$.

Now, in order to finish the proof it is sufficient to observe that if $d \neq h$ then either there will be a symbol $\bar{S}$ in an inner node or the deepest leaf (having the depth h) will not contain object a (because only d letters a will be propagated down). Hence, when c will arrive at this node, it will be transformed to $\#$.

Example 1. We present now an example and discuss the functioning of the system on it. Consider a system Π having 7 membranes with the following membrane structure:



Now consider the evolution of the system Π constructed as above. We represent it in a table format where each cell indicates the contents of the corresponding membrane at the given time moment. Since the evolution is non-deterministic, we consider firstly the correct evolution and after that we shall discuss unsuccessful cases.

Step	w_1	w_2	w_3	w_4	w_5	w_6	w_7
0	S_1						
1	S_2C_2	S	SC_1				
2	S_2b	Sa	$\bar{S}aC_2$	S	S	SC_1	
3	S_2bb	Saa	$S'a$	S	S	$\bar{S}C_2$	SC_1
4	S_2bbb	$Saaa$	$S'ba$	Sa	Sa	$\bar{S}a$	SC_1C_2
5	S_3bbb	$Saaac$	$S'bbc$	Saa	Saa	$S'a$	SC_3
6	S_3bb	$S'''aa$	$S''bb$	$Saac$	$Saac$	$S'bcC_3$	Sa
7	S_3b	$S'''a$	$S''bC_3$	$S'''a$	$S'''a$	$S''b$	Sac
8	S_3C_3	S'''	S''	S'''	S'''	S''	S'''
9	F	F	F	F	F	F	F

The system will fail in the following cases:

1. Signals C_1 and C_2 go to different membranes.
2. Some symbol $\bar{S}$ is not transformed to S' (or the deepest leaf does not contain a letter a).
3. S_3 appears in the root membrane after C_3 appears in a leaf.
4. The branch chosen by C_3 is not the longest (it has the depth d, $d < h$).

A possible evolution for the first unsuccessful case is represented in the table below:

Step	w_1	w_2	w_3	w_4	w_5	w_6	w_7
0	S_1						
1	S_2C_2	S	SC_1				
2	S_2b	SaC_2	$\bar{S}a$	S	S	SC_1	
3	S_2bb	$Saa\#$	$S'a$	S	S	$\bar{S}$	SC_1

A possible evolution for the second unsuccessful case is represented in the table below:

Step	w_1	w_2	w_3	w_4	w_5	w_6	w_7
0	S_1						
1	S_2C_2	S	SC_1				
2	S_2b	Sa	$\bar{S}aC_2$	S	S	SC_1	
3	S_2bb	Saa	$\bar{S}ba$	Sa	Sa	$\bar{S}aC_2$	SC_1
4	S_2bbb	$Saaa$	$\bar{S}bba$	Saa	Saa	$S'a$	SC_1C_2
5	S_3bbb	$Saaac$	$\bar{S}bbbc$	$Saaa$	$Saaa$	$S'ab$	aSC_3
6	S_3bb	$S'''aa$	$\#bbb$	$Saaac$	$Saaac$	$S'bbcC_3$	Saa

A possible evolution for the third unsuccessful case is represented in the table below:

Step	w_1	w_2	w_3	w_4	w_5	w_6	w_7
0	S_1						
1	S_2C_2	S	SC_1				
2	S_2b	Sa	$\bar{S}aC_2$	S	S	SC_1	
3	S_2bb	Saa	$S'a$	S	S	$\bar{S}C_2$	SC_1
4	S_2bbb	$Saaa$	$S'ba$	Sa	Sa	$\bar{S}a$	SC_1C_2
5	S_2bbbb	$Saaaa$	$S'bba$	Saa	Saa	$S'a$	aSC_3
6	S_3bbbb	$Saaaac$	$S'bbbc$	$Saaa$	$Saaa$	$S'baC_3$	Sa
7	S_3bbb	$S'''aaa$	$S''bbbC_3$	$Saaac$	$Saaac$	$S'bbc$	Saa
8	S_3bbC_3	$S'''aa$	$S''bb$	$S'''aa$	$S'''aa$	$S''bb$	$Saac$
9	$S_3b\#$	$S'''a$	$S''b$	$S''a$	$S'''a$	$S''b$	$S'''a$

A possible evolution for the fourth unsuccessful case is represented in the table below:

Step	w_1	w_2	w_3	w_4	w_5	w_6	w_7
0	S_1						
1	S_2C_2	S	SC_1				
2	S_2b	Sa	$\bar{S}aC_2$	S	SC_1	S	
3	S_2bb	Saa	$S'a$	S	SC_1C_2	$\bar{S}$	S
4	S_2bbb	$Saaa$	$S'ba$	Sa	SaC_3	$\bar{S}a$	S
5	S_3bbb	$Saaac$	$S'bbcC_3$	Saa	Saa	$S'a$	S
6	S_3bbC_3	$S'''aa$	$S''bb$	$Saac$	$Saac$	$S'bc$	Sa
7	$S_3b\#$	$S'''a$	$S''bC_3$	$S'''a$	$S'''a$	$S''b$	Sac

4 Deterministic case

Consider now the deterministic case. We take the class of P systems with promoters and inhibitors and solve Problem 1 for this class.

The idea of the algorithm is very simple. A symbol C_2 is propagated down to the leaves and at each step, being at a inner node, it sends back a signal C. At the root a counter starts to compute the height of the tree and it stop if and only if there are no more signals C. It is easy to compute that the last signal C will arrive at time $2h-1$ (there are h inner nodes, and the last signal will continue for $h-1$ steps). At the same time the height is propagated down the tree as in the non-deterministic case.

Below is the formal description of the system.

The P system $\Pi = (O, \mu, w_1, \cdots, w_n, R_1, \cdots, R_n)$ for deterministic synchronization is present below. We consider that μ is an arbitrary membrane structure. The set of objects is $O = \{S_1, S_2, S_3, S_4, S, \bar{S}, S', S'', S''', C_1, C_2, C, a, a', b, F\}$, the initial contents of the skin is $w_1 = \{S_1\}$, the other membranes are empty. The set of rules $R_1, \ldots, R_n$ are identical, they are presented below.

Start:

$$S_1 \to S_2; C_2'; S, in!; C_1, in! \tag{15}$$

Propagation of S:

$$S \to \bar{S}; S, in! \tag{16}$$

Propagation of C (height computing signal):

$$C_1 \to C_1, in! \qquad C_2 \to C; C_2, in!; C, out \tag{17}$$

$$C_1 C_2 \to \varepsilon \qquad C_2' \to C; C_2, in! \tag{18}$$

$$C \to C, out \tag{19}$$

Root counter:

$$S_2 \to S_3 \qquad S_3 \to S_3'; b; a, in! \mid_C \tag{20}$$

$$C \to \varepsilon \mid_{S_3} \qquad S_3' \to S_3 \mid_C \tag{21}$$

$$C \to \varepsilon \mid_{S_3'} \qquad S_3' \to S_4; a', in! \mid_{\neg C} \tag{22}$$

Propagation of a:

$$\bar{S}a \to S' \qquad a \to b; a, in! \mid_{S'} \tag{23}$$

End propagate of a:

$$a'S' \to S''; a', in! \qquad a'Sa \to S''' \tag{24}$$

Decrement:

$$S''b \to S'' \qquad S'''a \to S''' \tag{25}$$

$$S'' \to F\,|_{\neg b} \qquad S''' \to F\,|_{\neg a} \tag{26}$$

Root decrement:

$$S_4 b \to S_4 \qquad S_4 \to F\,|_{\neg b} \tag{27}$$

We now give a structural explanation of the system. Rule (15) produces four objects. Similar to the system from the previous section, the propagation of object S by (16) leads to marking the intermediate nodes by $\bar{S}$ and the leaves by S. While objects C_1, C_2 propagate down the tree structure and send a continuous stream of objects C up to the root by (17)-(19), object S_2 counts, producing by rules (20)-(22) an object b every other step.

When the counting stops, there will be exactly h copies of object b in the root. Similar to the construction from the previous section, objects a are produced together with objects b by the second rule from (20). Objects a are propagated down the structure and decremented by one at every level by (23).

After the counting stops in the root (the last rule from (22)), object a' is produced. It propagates down the tree structure by (24), leading to the appearance of objects S'' in the intermediate nodes and S''' in the leaves. These two objects perform the countdown and the corresponding nodes fire by (25). The root behaves in a similar way by (27).

The correctness of the construction can be argued as follows. It takes $h + 1$ steps for a symbol C_2 to reach all leaves. All this time, symbols C are sent up the tree. It takes further $h - 1$ steps for all symbols C to reach the root node, and one more step until symbols C disappear. Therefore, symbols b appear in the root node every odd step from step 3 until step $2h + 1$, so h copies will be made. Together with the production of b^h in the root node, this number propagates down the tree, being decremented by one at each level. For the depth i, the number $h - i$ is represented, during propagation, by the multiplicity of symbols a (one additional copy of a is made) in the leaves and by the multiplicity of symbols b in non-leaf nodes. After $2h + 2$ steps, the root node starts the propagation of the countdown (*i.e.*, decrement of symbols a or b). For a node of depth i, it takes i steps for the countdown signal (a') to reach it, another $h - i$ steps to eliminate symbols a or b, so every node fires after $2h + 2 + i + (h - i) + 1 = 3h + 3$ steps after the synchronization has started.

Example 2. Consider a P system having same membrane structure as the system from Example 1. We present below the evolution of the system in this case.

Step	w_1	w_2	w_3	w_4	w_5	w_6	w_7
0	S_1						
1	S_2C_2'	SC_1	SC_1				
2	S_3C	SC_1C_2	$\bar{S}C_2$	SC_1	SC_1	SC_1	
3	$S_3'bC$	Sa	$\bar{S}aC$	SC_1C_2	SC_1C_2	$\bar{S}C_2$	SC_1
4	S_3bC	Sa	$S'C$	S	S	$\bar{S}C$	SC_1C_2
5	$S_3'bbC$	Saa	$S'aC$	S	S	$\bar{S}$	S
6	S_3bbC	Saa	$S'b$	Sa	Sa	$\bar{S}a$	S
7	$S_3'bbb$	$Saaa$	$S'ba$	Sa	Sa	S'	S
8	S_4bbb	$a'Saaa$	$a'S'bb$	Saa	Saa	$S'a$	S
9	S_4bb	$S'''aa$	$S''bb$	$a'Saa$	$a'Saa$	$a'S'b$	Sa
10	S_4b	$S'''a$	$S''b$	$S'''a$	$S'''a$	$S'b$	$a'Sa$
11	S_4	S'''	S''	S'''	S'''	S'	S'''
12	F	F	F	F	F	F	F

5 Conclusions

In this article we presented two algorithms that synchronize two given classes of P systems. The first one is non-deterministic and it synchronizes the class of transitional P systems (with cooperative rules) in time $2h+3$, where h is the depth of the membrane tree. The second algorithm is deterministic and it synchronizes the class of P systems with promoters and inhibitors in time $3h+3$.

It is worth to note that the first algorithm has the following interesting property. After $2h$ steps either the system synchronizes and the object F is introduced, or an object $\#$ will be present in some membrane. This property can be used during an implementation in order to cut off failure cases.

The results obtained in this article rely on a rather strong target indication, $in!$, which sends an object to all inner membranes. It would be interesting to investigate what happens if such target is not permitted. However, we conjecture that a synchronization would be impossible in this case.

The study of the synchronization algorithms for different classes of P systems is important as it permits to implement different synchronization strategies which are important for such a parallel device as P systems. In particular, with such approach it is possible to simulate P systems with multiple global clocks by P systems with one global clock. It is particulary interesting to investigate the synchronization problem for P systems which cannot create new objects, for example for P systems with symport/antiport.

Acknowledgments. The first and the third authors acknowledge the support of the Science and Technology Center in Ukraine, project 4032.

Bibliography

[1] F. Bernardini, M. Gheorghe, M. Margenstern, S. Verlan, How to synchronize the activity of all components of a P system? In Gy. Vaszil (ed), *Proceedings of the International Workshop Automata for Cellular and Molecular Computing*, MTA SZTAKI, Budapest, Hungary, 2007, 11-22.

[2] E. Goto, A minimum time solution of the firing squad problem, *Course notes for applied mathematics*, **298**, (1962), Harvard University.

[3] J.B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, *Proceedings of the American Mathematical Society*, **7**, (1956), 48-50.

[4] J. Mazoyer, A six-state minimal time solution to the firing squad synchronization problem, *Theoretical Science*, **50**, (1987), 183-238.

[5] M. Minsky, *Computation: finite and infinite machines*, Prentice-Hall, 1967.

[6] Gh. Păun: *Membrane computing. An introduction*. Springer-Verlag, 2002.

[7] R.C. Prim, Shortest connection networks and some generalizations, *Bell System Technical Journal*, **36**, (1957), 1389-1401.

[8] PT. Spellman, G. Sherlock, Reply whole-cell synchronization - effective tools for cell cycle studies, *Trends in Biotechnology*, **22**(6), (2004), 270-273.

[9] H. Umeo, M. Maeda, N. Fujiwara, An efficient mapping scheme for embedding any one-dimensional firing squad synchronization algorithm onto two-dimensional arrays, **ACRI 2002**, *Lecture Notes in Computer Science*, **2493**, (2002), 69-81.

[10] H. Schmid, T. Worsch, The firing squad synchronization problem with many generals for one-dimensional CA, **IFIP TCS 2004**, (2004), 111-124.

[11] J.-B. Yunès, Seven-state solution to the firing squad synchronization problem, *Theoretical Computer Science*, **127**(2), (1994), 313-332.

[12] The P systems web page. `http://ppage.psystems.eu`.

(Tissue) P Systems Using Noncooperative Rules Without Halting Conditions

Markus Beyreder, Rudolf Freund

Vienna University of Technology, Faculty of Informatics,
Favoritenstr. 9, A-1040 Wien, Austria
{markus,rudi}@emcc.at

We consider (tissue) P systems using noncooperative rules, but considering computations without halting conditions. As results of a computation we take the contents of a specified output membrane/cell in each derivation step, no matter whether this computation will ever halt or not, eventually taking only results completely consisting of terminal objects only. The computational power of (tissue) P systems using noncooperative rules turns out to be equivalent to that of (E)0L systems.

1 Introduction

In contrast to the original model of P systems introduced in [5], in this paper we only consider noncooperative rules. Moreover, as results of a computation we take the contents of a specified output membrane in each derivation step, no matter whether this computation will ever halt or not, eventually taking only results completely consisting of terminal objects. In every derivation step, we apply the traditional maximal parallelism. Other derivation modes could be considered, too, but, for example, applying the sequential derivation mode would not allow us to go beyond context-free languages. As the model defined in this paper we shall take the more general one of tissue P systems (where the communication structure of the system is an arbitrary graph, e.g., see [4], [2]), which as a specific subvariant includes the original model of membrane systems if the communication structure allows for arranging the cells in a hierarchical tree structure.

The motivation to consider this specific variant of tissue P systems came during the Sixth Brainstorming Week in Sevilla 2008 when discussing the ideas presented in [3] with the authors Miguel Gutiérrez-Naranjo and Mario Pérez-Jiménez. They consider the evolution of deterministic (tissue) P systems with simple (i.e., noncooperative) rules and aim to find a mathematically sound representation of such systems in order to deduce their behavior and, on the other hand, to find suitable corresponding P systems for a given mathematical system with specific behavior. Whereas in that paper only deterministic P systems are considered, which allows for a mathematical representation

like for deterministic 0L systems, and as well real values for the coefficients assigned to the symbols are allowed, in this paper we restrict ourselves to the non-negative integer coefficients commonly used in traditional variants of (tissue) P systems.

We shall prove that the computational power of extended tissue P systems using noncooperative rules is equivalent to that of E0L systems when taking all results appearing in the specified output cell consisting of terminal objects only.

The present paper is organized as follows. Section 2 briefly recalls the notations commonly used in membrane computing and the few notions of formal language theory that will be used in the rest of the paper; in particular, we report the definition of (extended) Lindenmayer systems. Section 3 is dedicated to the definition of tissue P systems with noncooperative rules working in the maximally parallel derivation mode. The computational power of these classes of (extended) tissue P systems is then investigated in Section 4 in comparison with the power of the corresponding classes of (extended) Lindenmayer systems. Some further remarks and directions for future research are discussed in the last section.

2 Preliminaries

We here recall some basic notions concerning the notations commonly used in membrane computing (we refer to [6] for further details and to [9] for the actual state of the art in the area of P systems) and the few notions of formal language theory we need in the rest of the paper (see, for example, [8] and [1], as well as [7] for the mathematical theory of L systems).

An alphabet is a finite non-empty set of abstract symbols. Given an alphabet V, by V^* we denote the set of all possible strings over V, including the empty string λ. The length of a string $x \in V^*$ is denoted by $|x|$ and, for each $a \in V$, $|x|_a$ denotes the number of occurrences of the symbol a in x. A multiset over V is a mapping $M : V \longrightarrow \mathbb{N}$ such that $M(a)$ defines the multiplicity of a in the multiset M ($\mathbb{N}$ denotes the set of non-negative integers). Such a multiset can be represented by a string $a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)} \in V^*$ and by all its permutations, with $a_j \in V$, $M(a_j) \geq 0$, $1 \leq j \leq n$. In other words, we can say that each string $x \in V^*$ identifies a finite multiset over V defined by $M_x = \{(a, |x|_a) \mid a \in V\}$. Ordering the symbols in V in a specific way, i.e., $(a_1, \dots, a_n)$ such that $\{a_1, \dots, a_n\} = V$, we get a Parikh vector $\left(|x|_{a_1}, \dots, |x|_{a_n}\right)$ associated with x. The set of all multisets over V is denoted by M_V, the set of all Parikh vectors by $Ps\,(V^*)$. In the following, we shall not distinguish between multisets and the corresponding Parikh vectors. Given two multisets x and y, with $x, y \in V^*$, we say that the multiset x includes the multiset y, or the multiset y is included in the multiset x, and we write $x \sqsupseteq y$, or $y \sqsubseteq x$, if and only if $|x|_a \geq |y|_a$, for every $a \in V$. The union of two multisets x and y is denoted by $x \sqcup y$ and is defined to

be the multiset with $|x \sqcup y|_a = |x|_a + |y|_a$, for every $a \in V$. For $m, n \in \mathbb{N}$, by $[m..n]$ we denote the set $\{x \in \mathbb{N} \mid m \leq x \leq n\}$.

An extended Lindenmayer system (an E0L system for short) is a construct $G = (V, T, P, w)$, where V is an alphabet, $T \subseteq V$ is the *terminal* alphabet, $w \in V^*$ is the *axiom*, and P is a finite set of *noncooperative rules* over V of the form $a \to x$. In a derivation step, each symbol present in the current sentential form is rewritten using one rule arbitrarily chosen from P. The language generated by G, denoted by $L(G)$, consists of all the strings over T which can be generated in this way by starting from w. An E0L system with $T = V$ is called a 0L system. As a technical detail we have to mention that in the theory of Lindenmayer systems usually it is required that for every symbol a from V at least one rule $a \to w$ in P exists. If for every symbol a from V exactly one rule $a \to w$ in P exists, then this Lindenmayer system is called *deterministic*, and we use the notations DE0L and D0L systems. By $E0L$ and $0L$ ($DE0L$ and $D0L$) we denote the families of languages generated by (deterministic) E0L systems and 0L systems, respectively. It is known from [8] that $CF \subset E0L \subset CS$, with CF being the family of context-free languages and CS being the family of context-sensitive languages, and that CF and $0L$ are incomparable, with $\{a^{2^n} \mid n \geq 0\} \in D0L - CF$.

As the paper deals with P systems where we consider symbol objects, we will also consider E0L systems as devices that generate sets of (vectors of) non-negative integers; to this aim, given an E0L system G, we define the set of non-negative integers generated by G as the length set $N(G) = \{\, |x| \mid x \in L(G) \,\}$ as well as $Ps\,(G)$ to be the set of Parikh vectors corresponding to the strings in $L\,(G)$. In the same way, the length sets and the Parikh sets of the languages generated by context-free and context-sensitive grammars can be defined. The corresponding families of sets of (vectors of) non-negative integers then are denoted by NX and PsX, for $X \in \{E0L, 0L, DE0L, D0L, CF, CS\}$, respectively.

3 Tissue P Systems With Noncooperative Rules

Now we formally introduce the notion of tissue P systems with noncooperative rules by giving the following definition.

Definition 1 *An* extended tissue P system with noncooperative rules *is a construct*

$$\Pi = (n, V, T, R, C_0, i_0)$$

where

1. n *is the number of* cells*;*
2. V *is a finite alphabet of symbols called* objects*;*
3. $T \subseteq V$ *is a finite alphabet of* terminal symbols *(*terminal objects*);*

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

$$(a, i) \to (b_1, h_1) \dots (b_k, h_k)$$

for $i \in [1..k]$, $a \in V$ *as well as* $b_j \in V$ *and* $h_j \in [1..n]$, $j \in [1..k]$*;*
5. $C_0 = (w_1, \dots, w_n)$*, where the* $w_i \in V^*$*,* $i \in [1..n]$*, are finite multisets of objects for each* $i \in [1..n]$*,*
6. i_0 *is the* output cell.

A rule $(a, i) \to (b_1, h_1) \dots (b_k, h_k)$ in R_i indicates that a copy of the symbol a in cell i is erased and instead, for all $j \in [1..k]$, a copy of the symbol b_j is added in cell h_j.

In any configuration of the tissue P system, a copy of the symbol a in cell i is represented by (a, i), i.e., (a, i) is an element of $V \times [1..n]$.

Π is called *deterministic* if in every cell for every symbol from V exactly one rule exists.

¿From the initial configuration specified by $(w_1, ..., w_n)$, the system evolves by transitions getting from one configuration to the next one by applying a maximal set of rules in every cell, i.e., by working in the *maximally parallel derivation mode*. A *computation* is a sequence of transitions. In contrast to the common use of P systems to generate sets of multisets, as a result of the P system we take the contents of cell i_0, provided it only consists of terminal objects only, at each step of any computation, no matter whether this computation will ever stop or not, i.e., we do not take into account any halting condition, which in the following will be denoted by using the subscript u (for *unconditional halting*): the set of all multisets generated in that way by Π is denoted by $L_u(\Pi)$. If we are only interested in the number of symbols instead of the Parikh vectors, the corresponding set of numbers generated by Π is denoted by $N_u(\Pi)$.

The family of sets of multisets generated by tissue P systems with noncooperative rules with at most n cells in the maximally parallel derivation mode is denoted by $PsEOtP_n(ncoo, max, u)$ (u again stands for unconditional halting). Considering only the length sets instead of the Parikh vectors of the results obtained in the output cell during the computations of the tissue P systems, we obtain the family of sets of non-negative integers generated by tissue P systems with noncooperative rules with at most n cells in the maximally parallel derivation mode, denoted by $NEOtP_n(ncoo, max, u)$. The corresponding families generated by non-extended tissue P systems – where all symbols are terminal – are denoted by $XOtP_n(ncoo, max, u)$, $X \in \{Ps, N\}$. For all families generated by (extended) tissue P systems as defined before, we add the symbol D in front of t if the underlying systems are deterministic. If the number of cells is allowed to be arbitrarily chosen, we replace n by $*$.

3.1 A well-known example Consider the D0L system with the only rule $a \to aa$, i.e.,

$$G = (\{a\}, \{a\}, \{a \to aa\}, a).$$

As is well known, the language generated by G is $\left\{a^{2^n} \mid n \geq 0\right\}$ and therefore $N(G) = \{2^n \mid n \geq 0\}$.

The corresponding deterministic one-cell tissue P system is

$$\Pi = (\{a\}, \{a\}, \{(a,1) \to (a,1)(a,1)\}, (a)).$$

Obviously, we get $L_u(\Pi) = Ps(L(G))$ and $N(G) = N_u(\Pi)$.

We should like to point out that in contrast to this tissue P system without imposing halting, there exists no tissue P system with only one symbol in one cell

$$\Pi = (\{a\}, \{a\}, R, (w))$$

that with imposing halting is able to generate $\{2^n \mid n \geq 0\}$, because such systems can generate only finite sets (singletons or the empty set):

- if $w = \lambda$, then $N_u(\Pi) = \{0\}$;
- if R is empty, then $N_u(\Pi) = \{|w|\}$;
- if $w \neq \lambda$ and R contains the rule $a \to \lambda$, then $N_u(\Pi) = \{0\}$, because no computation can stop as long as the contents of the cell is not empty;
- if $w \neq \lambda$ and R is not empty, but does not contain the rule $a \to \lambda$, then R must contain a rule of the form $a \to a^n$ for some $n \geq 1$, yet this means that there exists no halting computation, i.e., $N_u(\Pi)$ is empty.

4 The Computational Power of Tissue P Systems With Noncooperative Rules

In this section we present some results concerning the generative power of (extended) tissue P systems with noncooperative rules; as we shall show, there is a strong correspondence between these P systems with noncooperative rules and E0L systems.

Theorem 1 *For all* $n \geq 1$,

$$\begin{aligned} PsE0L &= PsEOtP_n(ncoo, max, u) \\ &= PsEOtP_*(ncoo, max, u). \end{aligned}$$

Proof We first show that

$$PsE0L \subseteq PsEOtP_1 (ncoo, max, u) :$$

Let $G = (V, T, P, w)$ be an E0L system. Then we construct the corresponding extended one-cell tissue P system

$$\Pi = (1, V, T, R, (w), 1)$$

with

$$R = \{(a, 1) \to (b_1, 1) \dots (b_k, 1) \mid a \to b_1 \dots b_k \in P\}.$$

Due to the maximal parallel derivation mode applied in the extended tissue P system Π, the derivations in Π directly correspond to the derivations in G. Hence, $L(\Pi) = Ps(L(G))$.

As for all $n \geq 1$, by definition we have

$$PsEOtP_1 (ncoo, max, u) \subseteq PsEOtP_n (ncoo, max, u),$$

it only remains to show that

$$PsEOtP_* (ncoo, max, u) \subseteq PsE0L :$$

Let

$$\Pi = (n, V, T, R, (w_1, \dots, w_n), i_0)$$

be an extended tissue P system. Then we first construct the E0L system

$$G = (V \times [1..n], T_0, P, w)$$

with

$$w = \sqcup_{i=1}^{n} h_i (w_i)$$

($\sqcup$ represents the union of multisets) and

$$T_0 = h_{i_0} (T) \cup \cup_{j \in [1..n], j \neq i_0} h_j (V)$$

where the $h_i : V^* \to \{(a, i) \mid a \in V\}^*$ are morphisms with $h_i (a) = (a, i)$ for $a \in V$ and $i \in [1..n]$, as well as

$$P = R \cup P'$$

where P' contains the rule $(a, i) \to (a, i)$ for $a \in V$ and $i \in [1..n]$ if and only if R contains no rule for (a, i) (which guarantees that in P there exists at least one rule for every $b \in V \times [1..n]$).

We now take the projection $h : T_0^* \to T^*$ with $h((a, i_0)) = a$ for all $a \in T$ and $h((a, j)) = \lambda$ for all $a \in V$ and $j \in [1..n]$, $j \neq i_0$. Due to the direct correspondence of derivations in Π and G, respectively, we immediately obtain $Ps(h(L(G))) = L_u(\Pi)$.

As $E0L$ is closed under morphisms (e.g., see [8], vol. 1, p. 266f.) and therefore $L_u(\Pi) = Ps(L(G'))$ for some E0L system G', we finally obtain $L_u(\Pi) \in PsE0L$. □

As an immediate consequence of Theorem 1, we obtain the following results:

Corollary 1 *For all* $n \geq 1$,

$$\begin{aligned} NE0L &= NEOtP_n\left(ncoo, max, u\right) \\ &= NEOtP_*\left(ncoo, max, u\right). \end{aligned}$$

Proof Given an E0L system G, we construct the corresponding extended tissue P system Π as above in Theorem 1; then we immediately infer $N\left(G\right) = N_u\left(\Pi\right)$. On the other hand, given an extended tissue P system Π, by the constructions elaborated in Theorem 1, we obtain

$$N_u\left(\Pi\right) = N\left(G'\right) = \{|x| \mid x \in h\left(L\left(G\right)\right)\}$$

and therefore $N_u\left(\Pi\right) \in NE0L$. □

Corollary 2 *For* $X \in \{Ps, N\}$, $X0L = XOtP_1\left(ncoo, max, u\right)$.

Proof This result immediately follows from the constructions elaborated in Theorem 1 with the specific restriction that for proving the inclusion

$$PsOtP_1\left(ncoo, max, u\right) \subseteq Ps0L$$

we can directly work with the symbols of V from the given non-extended tissue P system Π for the 0L system G to be constructed (instead of the symbols from $V \times \{1\}$) and thus do not need the projection h to get the desired result $L_u\left(\Pi\right) = L\left(G\right) \in Ps0L$. Besides this important technical detail, the results of this corollary directly follow from Theorem 1 and Corollary 1, because any non-extended system corresponds to an extended system where all symbols are terminal. □

For tissue P systems with only one cell, the noncooperative rules can also be interpreted as antiport rules in the following sense: an antiport rule of the form a/x in a single-cell tissue P system means that the symbol a goes out to the environment and from there (every symbol is assumed to be available in the environment in an unbounded number) the multiset x enters the single cell. The families of Parikh sets and length sets generated by (extended, non-extended) one-cell tissue P systems using antiport rules of this specific form working in the maximally parallel derivation mode are denoted by $XEOtP_1\left(anti_{1,*}, max, u\right)$ and $XOtP_1\left(anti_{1,*}, max, u\right)$ for $X \in \{Ps, N\}$, respectively. We then get the following corollary:

Corollary 3 *For* $X \in \{Ps, N\}$,

$$XEOtP_1\left(anti_{1,*}, max, u\right) = XE0L$$

and

$$XOtP_1\left(anti_{1,*}, max, u\right) = X0L.$$

Proof The results immediately follow from the previous results and the fact that the application of an antiport rule $a/b_1 \ldots b_k$ has exactly the same effect on the contents of the single cell as the noncooperative evolution rule $(a,1) \rightarrow (b_1,1) \ldots (b_k,1)$. □

For one-cell tissue P systems, we obtain a characterization of the families generated by the deterministic variants of these systems by the families generated by the corresponding variants of Lindenmayer systems:

Corollary 4 *For* $X \in \{Ps, N\}$ *and* $Y \in \{ncoo, anti_{1,*}\}$,

$$XED0L = XEDOtP_1\left(Y, max\right)$$

and

$$XD0L = XDOtP_1\left(Y, max\right).$$

Proof As already mentioned in the proof of Corollary 2, the results immediately follow from the constructions elaborated in Theorem 1 with the specific restriction that for proving the inclusion $PsEDOtP_1\left(ncoo, max, u\right) \subseteq PsED0L$ we can directly work with the symbols of V from the given (extended) deterministic tissue P system Π for the ED0L system G to be constructed (instead of the symbols from $V \times \{1\}$) and thus do not need the projection h to get the desired result $L\left(\Pi\right) = L_u\left(G\right) \in PsED0L$. The remaining statements follow from these constructions in a similar way as the results stated in Corollaries 1, 2, and 3. □

The constructions described in the proofs of Corollary 2 and 4 cannot be extended to (non-extended, deterministic) tissue P systems with an arbitrary number of cells, because in that case again the application of a projection h would be needed.

5 Conclusions and Future Research

In this paper we have shown that the Parikh sets as well as the length sets generated by (extended) tissue P systems with noncooperative rules (without halting) coincide with the Parikh sets as well as the length sets generated by (extended) Lindenmayer systems.

In the future, we may also consider other variants of extracting results from computations in (extended) tissue P systems with noncooperative rules, for example, variants of halting computations or only infinite computations, as well as other derivation modes as the sequential or the minimally parallel derivation mode. For the extraction of results, instead of the intersection with a terminal alphabet we may also use other criteria like the occurrence/absence of a specific symbol.

As inspired by the ideas elaborated in [3], we may investigate in more detail the evolution/behavior of deterministic tissue P systems with noncooperative rules based on the mathematical theory of Lindenmayer systems: as there is a one-to-one correspondence between deterministic tissue P systems with noncooperative rules in one cell and D0L systems, the well-known mathematical theory for D0L systems can directly be used to describe/ investigate the behavior of the corresponding deterministic tissue P systems with noncooperative rules.

Acknowledgements. The authors gratefully acknowledge the interesting discussions with Miguel Gutiérrez-Naranjo and Mario Pérez-Jiménez on the ideas presented in their paper [3].

Bibliography

[1] J. Dassow and Gh. Păun, *Regulated Rewriting in Formal Language Theory*. EATCS Monograph in Theoretical Computer Science, Springer, Berlin, 1989.

[2] R. Freund, Gh. Păun, and M.J. Pérez-Jiménez, Tissue-like P systems with channel states. *Theoretical Computer Science* **330** (2005), 101–116.

[3] M.A. Gutiérrez-Naranjo and M.J. Pérez-Jiménez, Efficient computation in real-valued P systems, Proceedings Sixth Brainstorming Workshop On Membrane Computing (BWMC08), to appear.

[4] C. Martín-Vide, Gh. Păun, J. Pazos, and A. Rodríguez-Patón, Tissue P Systems, *Theoretical Computer Science*, **296** (2003), 295–326.

[5] Gh. Păun, Computing with Membranes, *Journal of Computer and System Sciences*, **61** (2000), 108–143.

[6] Gh. Păun, *Membrane Computing. An Introduction*. Natural Computing Series, Springer, Berlin, 2002.

[7] G. Rozenberg and A. Salomaa, *The Mathematical Theory of L Systems*. Academic Press, New York, 1980.

[8] G. Rozenberg and A. Salomaa (Eds), *Handbook of Formal Languages*. 3 volumes, Springer, Berlin, 1997.
[9] The P Systems Web Page: `http://ppage.psystems.eu`.

Modelling Ecosystems using P Systems: The Bearded Vulture, a case study

Mónica Cardona[1], M. Angels Colomer[1], Mario J. Pérez-Jiménez[2], Delfí Sanuy[3], Antoni Margalida[4]

[1]University of Lleida, Dpt. of Mathematics,
Av. Alcalde Rovira Roure, 191. 25198 Lleida, Spain
{mcardona,colomer}@matematica.udl.es

2University of Sevilla, Dpt. of Computer Science and Artificial Intelligence,
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
marper@us.es

[3]University of Lleida, Dpt. of Animal Production,
Av. Alcalde Rovira Roure, 191. 25198 Lleida, Spain
dsanuy@prodan.udl.cat

[4]Bearded Vulture Study & Protection Group
Adpo. 43 E-25520 El Pont de Suert (Lleida),Spain
margalida@inf.entorno.es

The Bearded Vulture (*Gypaetus barbatus*) is an endangered species in Europe that feeds almost exclusively on bone remains provided by wild and domestic ungulates. In this paper, we present a model of an ecosystem related with the Bearded Vulture in the Pyrenees (NE Spain), by using P systems. The evolution of six species are studied: the Bearded Vulture and other five subfamilies of domestic and wild ungulates that provide the bones they feed on. P systems provide a high level computational modelling framework which integrates the structural and dynamical aspects of ecosystems in a compressive and relevant way. P systems explicitly represent the discrete character of the components of an ecosystem by using rewriting rules on multisets of objects which represent individuals of the population and bones. The inherent stochasticity and uncertainty in ecosystems is captured by using probabilistic strategies. In order to give an experimental validation of the P system designed, we have constructed a simulator that allows us to analyse the evolution of the ecosystem under different initial conditions.

1 Introduction

Animal species are interconnected in a network in which some species depend other ones in terms of feeding [10], [26]. Variations of different biomass affects the composition of the population structures [24]. In mountain ecosystems, with a traditional

relationship between wild ungulates and their predators has been disrupted by the presence of domestic animals [6]. Animals located at the top of the ecological pyramid are susceptible to their presence and number. The abandonment of dead animals on the mountains is a major source of food for necrophagous species [15]. This is the case of the Bearded Vulture (*Gypaetus barbatus*) a threatened species which feeds on bone remains of domestic and wild ungulates.

The study of population ecology and how the species interact with the environment [13] is one of the aspects of the conservation biology of more interest for managers and conservationists [2]. A widespread tool used are the ecological models, which use mathematical representations of ecological processes [21].

In this study, we design a model that studies the evolution of an ecosystem located in the Pyrenees, taking advantage of the capacity the P Systems to work in parallel. P systems provide a high level computational modelling framework which integrates the structural and dynamical aspects of ecosystems in a compressive and relevant way. P systems explicitly represent the discrete character of the components of an ecosystem by using rewriting rules on multisets of objects which represent individuals of the population and biomass available. The inherent stochasticity and uncertainty in ecosystems is captured by using probabilistic strategies. The ecosystem included six species: Bearded Vulture as scavenger (predator) species and the Pyrenean Chamois (*Rupicapra pyrenaica*), Red Deer (*Cervus elaphus*), Fallow Deer (*Dama dama*), Roe Deer (*Capreolus capreolus*) and Sheep (*Ovis capra*) as carrion (prey) species. In order to give an experimental validation of the P system, designed we have constructed a simulator that allows us to analyse the evolution of the ecosystem under different initial conditions. The Bearded Vulture is an endangered species and so there are many projects that study its behaviour and how it is affected by its environment. Thanks to these studies there is a large amount of information available which is required to define the P System and to validate the results obtained.

The paper is structured as follows. In the next section, basic concepts of the ecosystem to be modelled are introduced. The most outstanding aspects of each species are detailed as well as the interactions among them. In Section 3, a dynamical probabilistic P system to describe the ecosystem is presented. In order to study the dynamics of the ecosystem, a simulator of that probabilistic P system is designed in Section 4. The following section is devoted to the analysis of the results produced by the simulator. Finally, conclusions are presented in the last section.

2 Modelling the Ecosystem

The ecosystem to be modelled is located in the Catalan Pyrenees, in the Northeast of Spain. This area contains a total of 35 breeding territories that constitutes 34.3% of the Bearded Vulture Spanish population in 2007 (n = 102). See Figure 1 [15].

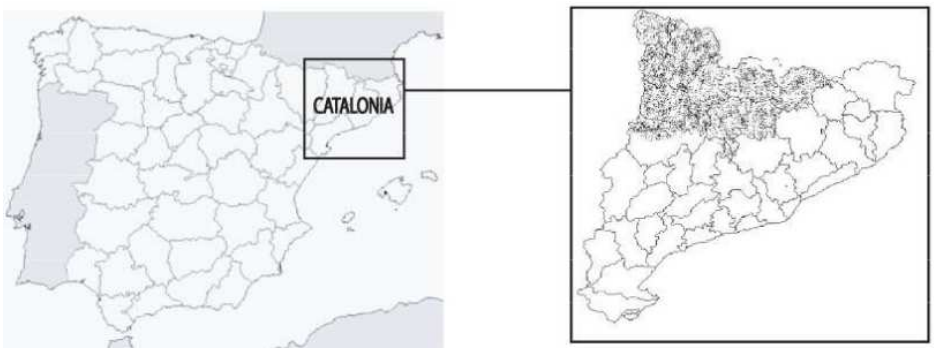


Fig. 2.1 Regional distribution of the Bearded Vulture in the Catalan Pyrenees (NE Spain).

Fig. 1. Regional distribution of the Bearded Vulture in the Catalan Pyrenees

The ecosystem to be modelled is composed of six species: the Bearded Vulture (predator species) and the Pyrenean Chamois, Red Deer, Fallow Deer, Roe Deer, and Sheep (prey species). Prey species belong to the bovid family, they are herbivores and their bone remains form the basic source of nourishment for the Bearded Vulture in the Pyrenees.

The Bearded Vulture is a cliff-nesting and territorial large scavenger distributed in mountains ranges in Eurasia and Africa. This is one of the rarest raptors in Europe (150 breeding pairs in 2007). This species has a mean lifespan in wild birds of 21.4 years [4]. The mean age of first breeding is 8.1 years, whereas the mean age of first successful breeding was 11.4 years [1]. Egg-laying take places during December-February and after 52-54 days of incubation and around 120 days of chick-rearing, the chick abandons the nest between June-August [19]. Clutch size in this species is usually of two eggs, but only one chick survives as a consequence of sibling aggression [18]. The female's annual fertility rate in Catalonia during the last five years is estimated around 38%.

The Bearded Vulture is the only vertebrate that feeds almost exclusively on bone remains. Its main food source is bone remains of dead small and medium-sized animals. In the Pyrenees, the remains of Pyrenean Chamois, Red Deer, Fallow Deer, Roe Deer, and Sheep form 67% of the vulture's food resources, and the rest 33% includes the remains of small size mammals (e.g., dogs, cats), large mammals (cows, horses), medium size mammals (e.g., wild boars) and birds [15]. A pair of Bearded Vultures needs an average 341 Kg of bones per year [17], [16].

In the first year, Bearded Vultures remain in the territory where they were born. During the dispersal period (from fledgling until the bird become territorial at 6–7 years), non-adult Bearded Vultures cover large distances surveying different areas. For example, the averaged surface covered by four youngs monitored after fledging was 4932 km^2

(range 950-10294 km^2, [23]). Breeding birds are territorial and the approximate home ranges obtained for eight pairs studied varied between 250 km^2 and 650 km^2. The average annual growth in the population of the Bearded Vultures in the Pyrenees has been estimated in 4-5%. The floating population principally remains in feeding stations situated in the central Pyrenees (Aragon).

The natural behaviour of the five bovid species is similar because they are all herbivores and they all reach the size of the adult animal when they are one year old. In general, they arrive at the sexual maturity within two years from birth. Pyrenean Chamois and the Red Deer have a longer life expectancy than Fallow Deer and Roe Deer (for a review of population parameters see [7], [8], [3], [9] and [20]. The natural mortality rates are similar in all five species, in the first year of life it is calculated to be 50% and 6% during there maining years. In spite of the great degree of similarity between these five species, there are differences between them, some are of natural origin and other are induced by human action. It is essential to bear them in mind in order to define a P system that can simulate the ecosystem in a reliable way.

Red Deer are appreciated very much by hunters, not for their meat but as a trophy and so only the males are hunted. This causes the natural evolution of the population to be modified. The hunter only takes the head as a trophy leaving the animal's body on the field, and so the carcass is eaten by other species and the bones may then be eaten by the Bearded Vulture.

Fallow Deer and Roe Deer live in areas that are difficult to reach and for this reason, the Bearded Vulture cannot take advantage of the bones of all of the dead animals.

As sheep [25] are domestic animals, humans exert a high level of control over their populations. The size and growth of the sheep population is limited by the owners of the flocks. The natural average life expectancy of sheep is longer that their actual life expectancy in the field because when its fertility rate decreases at the age of eight, they are taken out of the habitat. Most of the lambs are sold to market and so they are taken out of the habitat in the first year of life. Only 20%–30% of the lambs, mostly females, are left in the field and these are used to replace sheep that have died naturally and the old ones that have been removed from the flock. The number of animals in the Catalan Pyrenees the years 1994 and 2008 it is shown in Table 6.1 (see Appendix).

In this study, the feeding of the Bearded Vulture is dependent on the evolution of the P System. However the P System does not consider that the availability of food limits the feeding of the herbivorous, and so the growth of the vegetation is not modelled.

Taking all this background information into consideration, the following data was required for each species:

- I_1: age at which adult size is reached. Age at which the animal eats like the adult animal does. Moreover, at this age it will have surpassed the critical early phase during which the mortality rate is high;
- I_2: age at which it start to be fertile;
- I_3: age at which it stops being fertile;
- I_4: average life expectancy;
- I_5: fertility ratio (number of descendants by fertile female);
- I_6: mortality ratio in first years ($age < I_1$);
- I_7: mortality ratio in adult animals ($age \geq I_1$);
- I_8: percentage of females in the population.

The required information about each species is shown in Table 6.2 (see Appendix).

When an animal dies, the weight of bones which it leaves is around 20% of its total weight. Table 6.3 (see Appendix) shows the average weight of each animal as well as the weight of bones they leave. In the case of Fallow Deer and Roe Deer, the value of the weight of bones is then be multiplied by 0,2 (20%) which is the proportion of bones the Bearded Vulture may profit from.

In the P system, it is only considered the Bearded Vulture older than 8, because the younger ones are floating birds. There are seven feeding stations in Catalonia which provide around 10500 kg of bone remains annually. These artificial feeding sites have not been considered in the study and most of the floating birds feeds at them.

3 A P System Based Model of the Ecosystem

Membrane computing is a branch of Natural Computing that was initiated at the end of 1998 by Gh. Păun (by a paper circulated at that time on web and published in 2000 [22]). Since then it has received important attention from the scientific community. Details can be found at the web page `http://ppage.psystems.eu/`, maintained in Vienna under the auspices of the European Molecular Computing Consortium, EMCC.

In short, one abstracts computing models from the structure and the functioning of living cells, as well as from the organization of cell in tissues, organs, and other higher order structures. The main components of such a model are a cell-like *membrane structure*, in the *compartments* of which one places *multisets of symbol-objects* which evolve in a synchronous maximally parallel manner according to given *evolution rules*, also associated with the membranes.

The *semantic* of the P systems is defined as follows: a *configuration* of a P system consists of a membrane structure and a family of multisets of objects associated with

each region of the structure. At the beginning, there is a configuration called the *initial configuration* of the system.

In each time unit we can transform a given configuration in another one by applying the evolution rules to the objects placed inside the regions of the configurations, in a non–deterministic, and maximally parallel manner (the rules are chosen in a non–deterministic way, and in each region all objects that can evolve must do it). In this way, we get *transitions* from one configuration of the system to the next one.

A *computation* of the system is a (finite or infinite) sequence of configurations such that each ones is obtained from the previous one by a transition, and shows how the system is evolving. A computation which reaches a configuration where no more rules can be applied to the existing objects, is called a *halting computation*. The result of a halting computation is usually encoded by the multiset associated with a specific output membrane (or the environment) in the final configuration.

In this section, we present a model of the ecosystem described in Section 2 by means of probabilistic P systems. We will study the behaviour of this ecosystem under diverse initial conditions.

First, we define the P systems based framework (probabilistic P systems), where additional features such as two electrical charges which describe specific properties in a better way, are used.

Definition 1 *A probabilistic P system of degree n is a tuple*

$$\Pi = (\Gamma, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_n, R, \{c_r\}_{r \in R})$$

where:

- Γ *is the alphabet (finite and nonempty) of objects (the working alphabet);*
- μ *is a membrane structure, consisting of n membranes, labeled $1, 2, \ldots, n$. The skin membrane is labeled by 0. We also associate electrical charges with membranes from the set $\{0, +\}$, neutral and positive;*
- $\mathcal{M}_1, \ldots, \mathcal{M}_n$ *are strings over Γ, describing the multisets of objects initially placed in the n regions of μ;*
- R *is a finite set of evolution rules. An evolution rule associated with the membrane labelled by i is of the form $r: \; u[\, v\,]_i \xrightarrow{c_r} u'[\, v'\,]_i$, where u, v, u', v' are a multiset over Γ and c_r is a real number between 0 and 1 associated with the rule.*

We assume that a global clock exists, marking the time for the whole system (for all compartments of the system); that is, all membranes and the application of all rules are synchronized.

The n-tuple of multisets of objects present at any moment in the n *regions* of the system constitutes the *configuration* of the system at that moment. The tuple $(\mathcal{M}_1, \ldots, \mathcal{M}_n)$ is the initial configuration of the system.

The P system can pass from one configuration to another one by using the rules from R as follows: at each transition step, the rules to be applied are selected according to the probabilities assigned to them, and all applicable rules are simultaneously applied and all occurrences of the left–hand side of the rules are consumed, as usual.

3.1 The model Our model consists in the following probabilistic P system of degree 2 with two electrical charges (neutral and positive):

$$\Pi = (\Gamma, \mu, \mathcal{M}_1, \mathcal{M}_2, R, \{c_r\}_{r \in R})$$

where:

- In the alphabet Γ, we represent the six species of the ecosystem (index i is associated with the species and index j is associated with their age, and the symbols X, Y and Z represent the same animal but in different state); it also contains the auxiliary symbols B, which represents 0.5 kgs of bones, and C, which allows to change the polarization of membrane labelled by 2 in a specific stage.

$$\Gamma = \{X_{ij},\ Y_{ij},\ Z_{ij} :\ 1 \leq i \leq 7,\ 0 \leq j \leq k_{i,4}\} \cup \{\ B,\ C\}$$

- In the membrane structure, we represent two regions, the skin (where animals reproduce) and an inner membrane (where animals feed and die): $\mu = [\ [\ ]_2\ ]_1$ (neutral polarization will be omitted);
- In $\mathcal{M}_1$ and $\mathcal{M}_2$, we specify the initial number of objects present in each regions (encoding the initial population and the initial food);
  - $\mathcal{M}_1 = \{X_{ij}^{q_{ij}} : 1 \leq i \leq 7,\ 0 \leq j \leq k_{i,4}\}$, where the multiplicity q_{ij} indicates the number of animals, of species i whose age is j that are initially present in the ecosystem;
  - $\mathcal{M}_2 = \{C\ B^{\alpha}\}$, where α is defined as follows:

$$\alpha = \lceil \sum_{j=1}^{21} q_{1j} \cdot 1.10 \cdot 682 \rceil$$

    Value α represents an external contribution of food which is added during the first year of study so that the Bearded Vulture survives. In the formula, q_{1j} represents the number of j years of age Bearded Vultures, constant 1.10 represents 10% of the population growth and constant 682 represents the amount of food needed per year for a Bearded Vulture pair to survive.
- The set R of evolution rules consists of:

* Reproduction-rules.

  Adult males:

  - $r_0 \equiv [X_{ij} \xrightarrow{(1-k_{i,14})\cdot(1-k_{i,16})} Y_{ij}]_1,\ 1 \leq i \leq 7, k_{i,2} \leq j \leq k_{i,4}.$

  Adult females that reproduce:

  - $r_1 \equiv [X_{ij} \xrightarrow{k_{i,5}\cdot k_{i,14}\cdot(1-k_{i,16})} Y_{ij}Y_{i0}]_1,\ 1 \leq i \leq 7,\ k_{i,2} \leq j < k_{i,3}.$

  Adult females that do not reproduce:

  - $r_2 \equiv [X_{ij} \xrightarrow{(1-k_{i,5})\cdot k_{i,14}\cdot(1-k_{i,16})} Y_{ij}]_1,\ 1 \leq i \leq 7,\ k_{i,2} \leq j < k_{i,3}.$

  Young animals that do not reproduce:

  - $r_3 \equiv [X_{ij} \xrightarrow{1-k_{i,16}} Y_{ij}]_1,\ 1 \leq i \leq 7,\ 0 \leq j < k_{i,2}.$

* Growth rules.
  - $r_4 \equiv [X_{ij} \xrightarrow{(k_{i,6}+k_{i,9})\cdot k_{i,16}} Y_{ik_{i,2}}Y_{ij}]_1,\ 1 \leq i \leq 7,\ k_{i,2} \leq j < k_{i,4}.$
  - $r_5 \equiv [X_{ij} \xrightarrow{k_{i,6}\cdot k_{i,16}} Y_{ik_{i,2}}Y_{ij}]_1,\ 1 \leq i \leq 7,\ j = k_{i,4}.$
  - $r_6 \equiv [X_{ij} \xrightarrow{(1-k_{i,6}-k_{i,9})\cdot k_{i,16}} Y_{ij}]_1,\ 1 \leq i \leq 7,\ k_{i,2} \leq j \leq k_{i,4}.$

* Young animals mortality rules.

  Those which survive:

  - $r_7 \equiv Y_{ij}[\,]_2 \xrightarrow{1-k_{i,7}-k_{i,8}} [Z_{ij}]_2 :\ 1 \leq i \leq 7,\ 0 \leq j < k_{i,1}.$

  Those which die and leaving bones:

  - $r_8 \equiv Y_{ij}[\,]_2 \xrightarrow{k_{i,8}} [B^{k_{i,12}}]_2 :\ 1 \leq i \leq 7,\ 0 \leq j < k_{i,1}.$

  Those which die and do not leave bones:

  - $r_9 \equiv Y_{ij}[\,]_2 \xrightarrow{k_{i,7}} [\,]_2 :\ 1 \leq i \leq 7,\ 0 \leq j < k_{i,1}.$

* Adult animals mortality rules.

  Those which survive:

  - $r_{10} \equiv Y_{ij}[\,]_2 \xrightarrow{1-k_{i,9}-k_{i,10}} [Z_{ij}]_2 :\ 1 \leq i \leq 7,\ k_{i,1} \leq j < k_{i,4}.$

  Those which die leaving bones:

  - $r_{11} \equiv Y_{ij}[\,]_2 \xrightarrow{k_{i,10}} [B^{k_{i,13}}]_2 :\ 1 \leq i \leq 7,\ k_{i,1} \leq j < k_{i,4}.$

  Those which die and do not leave bones:

  - $r_{12} \equiv Y_{ij}[\,]_2 \xrightarrow{k_{i,9}} [\,]_2 :\ 1 \leq i \leq 7,\ k_{i,1} \leq j < k_{1,4}.$

  Animals that die at an average life expectancy:

  - $r_{13} \equiv Y_{ij}[\,]_2 \xrightarrow{1-k_{i,16}} [B^{k_{i,11}\cdot k_{i,13}}]_2 :\ 1 \leq i \leq 7,\ j = k_{i,4}.$

- $r_{14} \equiv Y_{ij}[\,]_2 \xrightarrow{k_{i,16}} [Z_{ik_{i,2}}]_2 : \ 1 \leq i \leq 7, \ j = k_{i,4}$.

* Feeding rules.
  - $r_{15} \equiv [Z_{ij}B^{k_{i,15}}]_2 \rightarrow X_{ij+1}[\,]_2^+ : \ 1 \leq i \leq 7, \ 0 \leq j \leq k_{i,4}$.
* Rules of mortality due to a lack of food, and the elimination of those bones that are not eaten by the Bearded Vulture from the system.

  Elimination of remaining bones:
  - $r_{16} \equiv [B]_2^+ \rightarrow [\,]_2$.
  - $r_{17} \equiv [C]_2^+ \rightarrow [C]_2$.

  Adult animals that die because they have not enough food:
  - $r_{18} \equiv [Z_{ij}]_2^+ \rightarrow [B^{k_{i,11} \cdot k_{i,13}}]_2 : \ 1 \leq i \leq 7, \ k_{i,1} \leq j \leq k_{i,4}$

  Young animals that die because they have not enough food:
  - $r_{19} \equiv [Z_{ij}]_2^+ \rightarrow [B^{k_{i,11} \cdot k_{i,12}}]_2 : \ 1 \leq i \leq 7, \ j < k_{i,1}$

The constants associated with the rules have the following meaning:

- $k_{i,1}$: age at which adult size is reached. This is the age at which the animal eats like the adult does, and at which if the animal dies, the amount of biomass it leaves is similar to the total one left by an adult. Moreover, at this age it will have surpassed the critical early phase during which the mortality rate is high.
- $k_{i,2}$: age at which it starts to be fertile.
- $k_{i,3}$: age at which it stops being fertile.
- $k_{i,4}$: average life expectancy.
- $k_{i,5}$: fertility ratio (number of descendants by fertile female).
- $k_{i,6}$: population growth.
- $k_{i,7}$: mortality ratio in first years ($age < k_{i,1}$) in which biomass in the form of bones is not left on the field.
- $k_{i,8}$: mortality ratio in first years ($age < k_{i,1}$) in which biomass in the form of bones is left on the field.
- $k_{i,9}$: mortality ratio in adult animals ($age \geq k_{i,1}$) in which biomass in the form of bones is not left on the field.
- $k_{i,10}$: mortality ratio in adults animals ($age \geq k_{i,1}$) in which biomass in the form of bones is left on the field.
- $k_{i,11}$ is equal to 1 if the animal dies at the age of $k_{i,4}$ leaving biomass, and it is equal to 0 if the animal dies at the age of $k_{i,4}$ without leaving bones.

- $k_{i,12}$: amount of bones from young animals ($age < k_{i,1}$).
- $k_{i,13}$: amount of bones from adult animals ($age \geq k_{i,1}$).
- $k_{i,14}$: percentage of females in the population.
- $k_{i,15}$: amount of food necessary per year and breeding pair (1 unit is equal 0.5 kg of bones).
- $k_{i,16}$: it is equal to 0 when the species go through a natural growth (animals which remain in the same territory throughout their lives) and it is equal to 1 when animals are nomadic (the Bearded Vulture moves from one place to another until it is 6–7 years old, when it settles down).

Besides, values for each species are shown in Table 6.4 (see Appendix). Most values in that table are equal to those in Table 6.2 (see Appendix), but it is necessary to make a remark on values $k_{4,10}$ and $k_{1,15}$. Value $k_{4,10}$ is obtained by adding 6% of natural mortality to 30% of animals killed by hunters. Value $k_{1,15}$ is 67% of 682 units ($341 \cdot 2$) which is the feeding the Bearded Vulture obtains from the other five species of ungulates modellized in the ecosystem.

The P system designed implements a four–stage–running. The first one is devoted to the reproduction of the diverse species in the ecosystem. Then, the animals mortality is analyzed according to different criteria. The third stage analyzes the amount of food in the ecosystem. In the last stage, the removal of animals because of a lack of food takes place. These stages are depicted in Figure 2.

Fig. 2. Structure of the P system running

4 A Simulator

In order to study the dynamics of the species that belong to the ecosystem, we have designed a simulator written in $C++$ language. This program runs on a PC.

In the simulation, the objects that encode the species and the age are represented by two vectors which are related through the number assigned to each animal of the ecosystem. The objects of the P system evolve in a random way; this stochasticity is implemented by generating random numbers between 1 and 100, according to an uniform distribution. One of the generated numbers is assigned to each animal. Then, the animal evolves according to the assigned number and the constant probability. For example, when the probability of surviving is 70%, the animal will die if the assigned number is higher than 70.

The input of the program consists of the parameters of each species that are considered in the P system and the number of animals of each species and age that are present at time zero. The output is the number and age of animals of each species that are present every year after completing the following processes: reproduction, mortality and feeding.

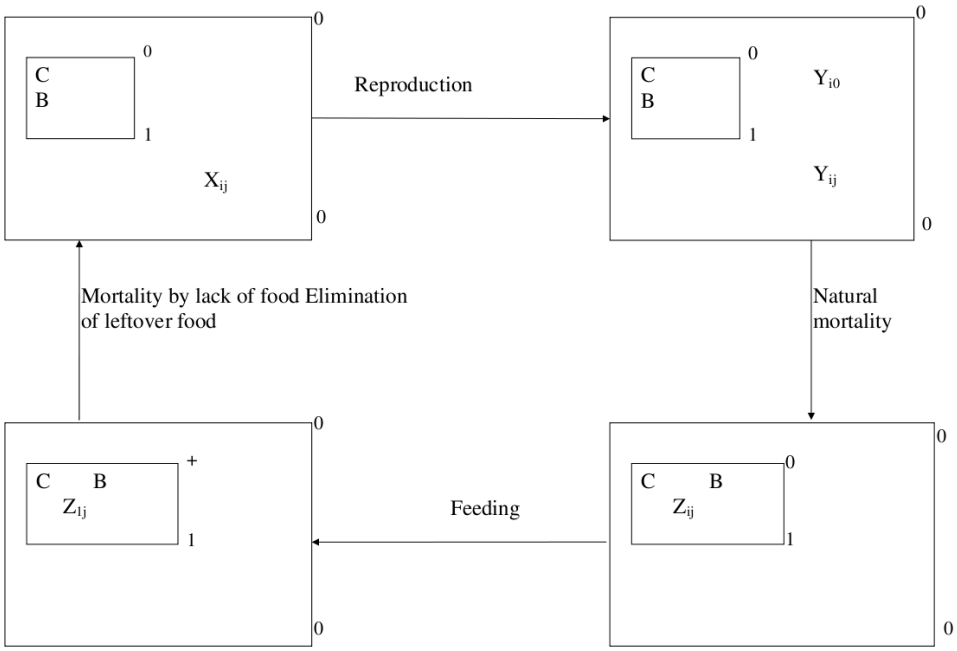


Fig. 3.2 Schema of the P system

In nature, an ecosystem is governed by nondeterminism, and this implies a complex mathematical model. Nevertheless, all the processes that are carried out have an important degree of randomness. This randomness can be predicted and can be quantified at every moment and situation of the ecosystem.

The program has been structured in four modules which correspond to each of the stages in which the P system is implemented.

- *Reproduction.* The inputs are the age at which each species begins to be fertile, the age at which it stops being fertile, the fertility rate, and the proportion of females of the species. This module also requires the total number of existing animals and the distribution of these animals in terms of species and ages. The output of this module is the number and age of animals of each species.
  The population growth of the Bearded Vulture is not obtained from the natural reproduction of the animals in the ecosystem but it depends on the floating population and the environment.
  The annual growth ratio has been obtained by R. Heredia [11] in a experimental way. So that, another input of this module is the growth percentage with respect to the total population, and the output (as in the case of animals natural reproduction) is the number of animals at each age.

- *Mortality*. The inputs are the mortality rate based on the age, the average life expectancy of each species, and finally the weight of bones left by the dead animal which is dependent on its age. Once again, this module also requires the total number of animals and their distribution in terms of species and ages. As with the other modules, the output of this module is the number and age of animals of each species when the process is completed. Another output of this module is the amount of food that is generated in terms of the weight of bones produced that provide the Bearded Vulture's basic source of nourishment.
- *Feeding*. The inputs are the amount of food available in the ecosystem and the annual amount of food that is necessary for the animal to survive under suitable conditions, in other words, conditions under which the animals are not debilitated and do not suffer the consequent effects on their capabilities. As was seen in the previous modules, inputs are generated by the P system itself as it quantifies objects representing the number of animals of each existing species and age. Once again, the output of this module is the number and age of animals of each species.
- *Elimination* of unused leftover food and the animal mortality from insufficient feeding. The input of this module is part of output of the feeding module. The aim of this process is to eliminate the number of animals that were not able to find the necessary amount of food for their survival, and also to consider the amount of leftover food that is degraded with time and that therefore stops having a role in the model. The animals that die due to a lack of food are transformed into bones that can then be eaten by the Bearded Vulture. The output of this module is an amount of food in form of bones that is available to the Bearded Vulture.

The unit of reference used in this study is the year, that is, the food consumed throughout an annual period is given at one single point in time, and with one application of each rule. The mortality of animals in an ecosystem is also a process that is carried out in a continuous way, throughout the year. However, reproduction is an activity that takes place at a specific time of the year, and moreover, it takes place at the same time for all of the species considered in this study. It will be necessary to verify if the one year unit of time chosen is correct or whether a shorter unit of time should be used in the P system. It is also necessary to check the robustness of the proposed model and to do this, it is run a second time with a modified order of application of the four processes modules. Given independence of the four modules that form the P system, it would be a simple exercise to run probability experiments with each module.

5 Results and Discussion

We have run our simulations using a program written in C++ language incorporating a specification of our model. We have considered the year as unit of time, so it has been necessary to discretize feeding and mortality variables.

As shown in Table 6.1 (see Appendix), data about the current number of animals in the Catalan Pyrenees do not specify the ages of animals. An age distribution has been estimated considering the different constants that affect the animals throughout their life. These constants are fertility rate, mortality rate and percentage of females in the population. We have obtained two estimations, one for the year 1994 which has been used for the experimental validation, shown in Table 6.5 (see Appendix), and another for the year 2008 which has been used for study the robustness of the P system, shown in Table 6.6 (see Appendix).

5.2 Robustness First, we have studied the robustness of our P system model with respect to some parameters.

According to the design of the P system, reproduction rules have a higher priority than mortality ones. Then, the robustness of the model regarding the change of that priority is analyzed. For that reason, two variants of the simulator have been studied changing the order of the corresponding modules. This fact can be implemented in the P system by changing variable X by variable Y in the initial multiset $\mathcal{M}_1$.

In both cases, the simulator was ran 10 times until it covered a period of 20 years, being the input the number of animals in 2008.

In Figure 3, solid lines and dashes lines represent the population dynamics when the simulator modules are applied following the *orders* reproduction–mortality–feeding and mortality–feeding–reproduction, respectively. Taking into account that the P system behaviour is similar in both cases, it can be deduced that our model is robust with regard to the properties considered.

5.3 Experimental Validation Let us suppose that we are studying a phenomenon of which we have (enough amount of) data experimentally obtained (at laboratory, through field–work, etc.) from some prefixed conditions. Let us suppose that we design a computational device trying to capture the most relevant facts of it, and we have a program which allows us to run simulations. We can say the model is experimentally validated if the results obtained with the simulator (from initial configurations corresponding to the prefixed conditions) are in agreement with the experimental data.

Bearing in mind that Table 6.1 (see Appendix) shows those data experimentally obtained corresponding to the years from 1994 to 2008, (being the input the number of animals in 1994) until it covered a period of 14 years. We have run our simulator 10 times, because it supposes a reduction of 70% of the deviation.

Table 6.7 (see Appendix) and Figure 4 show the difference between the average number of animals species obtained with the simulator compared with the censures estimate for 2008. In 2004, the Pyrenean Chamois species was affected by a disease which made the number of animals decrease to 10000. In the third column of that table, the evolution of

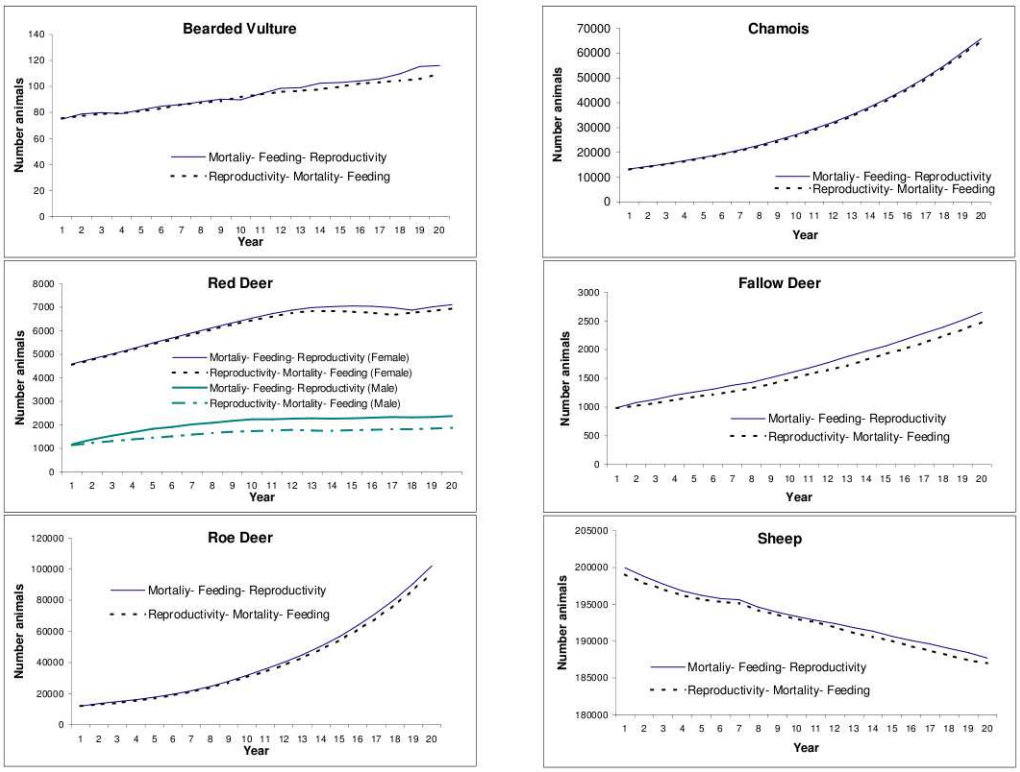


Fig. 5.3 Robustness of the ecosystem

the P system is shown without taking into account this piece of information, while in the fourth column it has been considered.

The P system proposed can be taken as a good model to study the evolution of an ecosystem. Variations noticed among the available data about the number of animals of each species from 1994 to 2008 (see Table 6.1) (see Appendix), are almost of no importance if we take into account that these data are taken from estimated census and they are never exact. Under the same conditions as starting point, the ecosystem has a certain behaviour pattern as it evolves, showing variations inherent to probabilistic systems.

The very important factor of population density was not considered in the model of the ecosystem. In this sense, as has been documented in other raptor species, density dependence and environmental stochasticity are both potentially important processes influencing population demography and long-term population grow [12]. For this reason why the population of some of the species such as Roe Deer, Fallow Deer and Chamois may grow in an exponential way reaching values which cannot be obtained

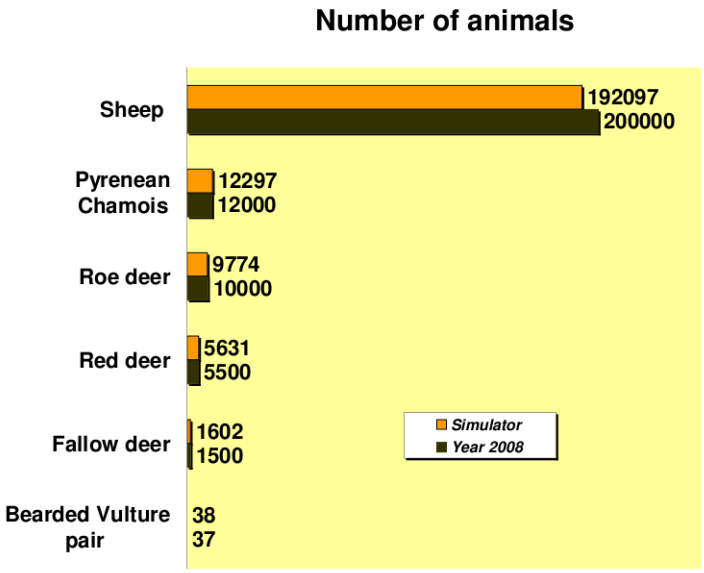


Fig. 5.4 Average number of animals

in the ecosystem. It is well–known, for example, that when a population of Red Deer reaches a level of 15000 animals, a regulation process starts that implies a drastic decrease of the population down to 1000 individuals. So that, if these factors are not taken into account, it may not be suitable for the study of the ecosystem dynamics in the long term.

Neither ungulates feeding nor the population density have been taken into account. This implies an exponential growth of ungulate species which constitute the basic source of feeding for the Bearded Vulture. Consequently, there is a continuous growth in the number of pairs of Bearded Vultures. According to some researches [14], the estimated maximum number of pairs of Bearded Vultures within the area under study is about fifty. Higher numbers of pairs would lead to competition among them and as subsequent decrease in the population down to values which the ecosystem can accept.

6 Conclusions and Future Work

A probabilistic P System which models an ecosystem related with the Bearded Vulture, that is located in the Catalan Pyrenees, has been presented.

By using this P System, it has been possible to study the dynamics of the ecosystem modifying the framework in order to analyze how the ecosystem would evolve if different biological factors were modified either by nature or through human intervention.

A simulator of the P System has been designed and the robustness of the model with respect to the order of application of different kinds of rules, has been shown.

Since the P System does not consider levels of population density, an exponential growth of populations of species is obtained. In a future work, this factor and other parameters (i.e. the amount of food of the hervibores species, the climatic changes in the ecosystem, etc.) should be considered.

In order to obtain a model which allow us to study the evolution of an ecosystem in the long term, it is necessary to take into account certain biological factors such as the following:

- Maximum population density for each species.
- Available feeding in the area on which the ungulates may feed.
- Amount of food daily eaten by each of the ungulate species regarding their age.

Moreover, under adequate environmental conditions, the especies has a certain behaviour so that some values of the biological parameters can be accepted. When essential environmental conditions such as temperature and rainfall are not the adequate ones, biological constants change as a reaction to the environment. It can be accepted a model based on Markov chains in order to model temperature and rainfall. P systems modelling Markov chains were previously presented in [5] and they should be considered in order to improve some results.

Acknowledgements. M.J. Pérez-Jiménez wishes to acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, co–financed by FEDER funds, and of the Project of Excelence TIC 581 of the Junta de Andalucia.

Financial support for A.Margalida was obtained from the Departament de Medi Ambient i Habitatge of Generalitat de Catalunya.

Bibliography

[1] R.J. Antor, A. Margalida, H. Frey, R. Heredia,L. Lorente, J.A. Ses. Age of first breeding in wild and captive populations of Bearded Vultures (Gypaetus barbatus). *Acta Ornithologica*, **42** (2007), 114–118.

[2] M. Begon, J.L. Harper, C.R. Townsend. *Ecology: Individuals, Populations and Communities*. Blackwell Scientific Publications Inc., Oxford, UK, 1988.

[3] F. Braza, C. San Jos, A. Blom, V. Cases, J. E. Garca. Population parameters of fallow deer at Doana National Park (SW Spain). *Acta Theriol*, **35** (1990), 277–288

[4] C.J.Brown. Population dynamics of the bearded vulture Gypaetus barbatus in southern Africa. *African Journal of Ecology*, **35** (1997), 53–63.

[5] M. Cardona, M.A. Colomer, M.J. Prez-jimnez, A. Zaragoza. Handling Markov chains with membrane computing. *Lecture Notes in Computer Science*, **4135** (2006), 72–85.

[6] C. Chocarro, R. Fanlo, F. Fillat, P. Marn. Historical Evolution of Natural Resource Use in the Central Pyrenees of Spain. *Mountain Research And Development* , **10** (1990), 257–265.

[7] T. Clutton-Brock, F. E. Guinness, S. D. Albon. *Red deer: Behavior and Ecology of Two Sexes.* Edinburgh University Press, Edinburgh, 1982.

[8] R. Garca-Gonzlez, J. Herrero, R. Hidalgo. Estimacin puntual de diversos parmetros poblacionales y distributivos del sarrio en el Pirineo Occidental. *Pirineos*, **35** (1985), 53–63.

[9] I. Garin, J. Herrero. Distribution, abundance and demographic parameters of the Pyrenean Chamois (Rupicapra p. pyrenaica) in Navarre, Western Pyreness. *Mammalia*, **61** (1997), 55–63.

[10] P. H. Harvey, A. Purvis. Understanding the ecological and evolutionary reasons for life history variation: mammals as a case study. In McGlade J (ed) *Advanced ecological theory: principles and applications.* Blackwell Science Publications, Oxford, 1999, pp. 232–247.

[11] R. Heredia. Status y distribucin del quebrantahuesos en Espaa y diagnstico de la situacin de la poblacin en la UE. In A. Margalida and R. Heredia, eds. *Biologa de la conservacin del quebrantahuesos Gypaetus barbatus en Espaa.* Madrid: Organismo Autnomo Parques Nacionales, 2005.

[12] O. Krger. Long-term demographic analysis in goshawk *Accipiter gentilis*: the role of density dependence and stochasticity. *Oecologia*, **152**, (2008): 459–471.

[13] R. Margalef. *Ecologa.* Universidad Nacional de Educacin a Distancia, Madrid, 1977.

[14] A. Margalida, J.A. Donzar, J. Bustamante, F.J. Hernndez, M. Romero-Pujante. Application of a predictive model to detect long-term changes in nest-site selection in the Bearded Vulture Gypaetus barbatus : conservation in relation to territory shrinkage. *Ibis*, **150**, (2008), 242–249.

[15] A. Margalida, D. Garca, A. Corts-Avizanda. Factors influencing the breeding density of Bearded Vultures, Egyptian Vultures and Eurasian Griffon Vultures in Catalonia (NE Spain): management implications. *Animal Biodiversity and Conservation*, **30**, 2 (2007), 189–200.

[16] A. Margalida, S. Maosa, J. Bertran, D. Garca. Biases in Studying the Diet of the Bearded Vulture. *The Journal of Wildlife Management*, **71**, 5 (2006), 1621–1625.

[17] A. Margalida, J. Bertran, J. Boudet. Assessing the diet of nestling Bearded Vultures: a comparison between direct observation methods, *Journal of Field Ornithology*, **76**, 1 (2005), 40–45.

[18] A. Margalida, J. Bertran, J. Boudet, R. Heredia. Hatching asynchrony, sibling aggression and cannibalism in the Bearded Vulture (Gypaetus barbatus). *Ibis*, **146** (2004), 386–393.

[19] A. Margalida, D. Garca, J. Bertran, R. Heredia. Breeding biology and success of the Bearded Vulture Gypaetus barbatus in the eastern Pyrenees. *Ibis*, **145** (2003),

244–252.
[20] P. Mateos-Quesada, J. Carranza. Reproductive patterns of roe deer in central Spain. *Etologa*, **8** (2000), 9–12.
[21] H. McCallum. *Population parameters: estimation for ecological models*, Blackwell Science Publications, Oxford, 2000.
[22] Gh. Păun, Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for Computer Science-TUCS Report* Nr. 208, 1998.
[23] C. Sunyer. El periodo de emancipacin en el Quebrantahuesos: consideraciones sobre su conservacin. In R. Heredia, B. Heredia, (Eds.) *El Quebrantahuesos (Gypaetus barbatus) en los Pirineos* Coleccin Tcnica. Madrid: ICONA, 1991, pp. 47–65.
[24] D. Tilman. *Resource Competition and Community Structure*. Princeton University Press, Princeton, New Jersey, 1982.
[25] R. Torres. *Conservacin de recursos genticos ovinos en la raza Xisqueta: caracterizacin estructural, racial y gestin de la diversidad en programas "in situ"*. Ph D Thesis. Universitat Autnoma de Barcelona, Barcelona, 2006.
[26] A. Watson. *Animal Populations in Relation to Their Food Resources*. Blackwell Scientific Publications, Oxford, 1970.

Appendix

Table 6.1 Number of animals in the Catalan Pyrenees

Specie	1994	2008
Bearded Vulture pairs	20	37
Pyrenean Chamois	9000	12000
Red deer	1000	5500
Fallow deer	600	1500
Roe deer	1000	10000
Sheep	15000	200000

Table 6.2 Natural constants used in the model

Species	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
Bearded Vulture	1	8	20	21	38	6	12	50
Pyrenean Chamois	1	2	18	18	75	60	6	55
Red Deer	1	2	17	17-20	75	34	6	50
Fallow Deer	1	2	12	12	55	50	6	75
Roe Deer	1	1	10	10	100	58	6	67
Sheep	1	2	8	8	75	15	3	96

Table 6.3 Descriptive variables used to model the ecosystem

Specie	Weigh Male kg	Weigh Female kg	Percentage Female	Average weigh kg	Biomass: bone adult kg	Biomass: bone young kg	Kg accessible by B. Vulture (adult/young)
Bearded Vulture	5	6.5	60	5.75	-	-	-
Chamois	28	32	50	30	6	3	6/3
Red Deer Female	-	75	-	75	15	7.5	15/7.5
Red Deer Male	120	-	-	120	24	12	24/12
Fallow Deer	63	42	80	46	9	4.5	2/1
Roe Deer	27	23	66	24	5	2.5	1/0.5
Sheep	42	35	97	35.2	7	3.5	7/3.5

Table 6.4 Constants used in the P system based model

Specie	i	$k_{i,1}$	$k_{i,2}$	$k_{i,3}$	$k_{i,4}$	$k_{i,5}$	$k_{i,6}$	$k_{i,7}$	$k_{i,8}$	$k_{i,9}$	$k_{i,10}$	$k_{i,11}$	$k_{i,12}$	$k_{i,13}$	$k_{i,14}$	$k_{i,15}$	$k_{i,16}$
Bearded Vulture	1	1	8	20	21	-	4	6	0	12	0	0	0	0	50	460	1
Pyrenean Chamois	2	1	2	18	18	75	-	0	60	0	6	1	6	12	55	-	0
Red Deer Female	3	1	2	17	17	75	-	0	34	0	6	1	15	30	100	-	0
Red Deer Male	4	1	2	-	20	-	-	0	34	0	36	1	24	48	0	-	0
Fallow Deer	5	1	2	12	12	55	-	0	50	0	6	1	2	4	75	-	0
Roe Deer	6	1	1	10	10	100	-	0	58	0	6	1	1	2	67	-	0
Sheep	7	1	2	8	8	75	-	57	15	0	3	0	7	14	96	-	0

Table 6.5 Estimation of number of animals per age in 1994

Age	Bearded Vulture	Chamois	Red deer female	Red deer male	Fallow deer	Roe deer	Sheep
1	0	741	167	58	83	121	20832
2	0	740	133	44	73	121	20208
3	0	668	107	35	69	121	19601
4	0	667	85	28	63	121	19014
5	0	667	68	23	59	109	18443
6	0	596	41	14	55	108	17890
7	0	594	33	11	51	108	17353
8	2	518	26	9	47	96	16659
9	2	517	21	7	35	96	0
10	2	444	17	5	33	0	0
11	2	444	13	5	30	0	0
12	2	444	11	4	0	0	0
13	2	373	9	3	0	0	0
14	1	373	7	2	0	0	0
15	1	372	5	2	0	0	0
16	1	296	4	1	0	0	0
17	1	296	3	1	0	0	0
18	1	252	0	0	0	0	0
19	1	0	0	0	0	0	0
20	1	0	0	0	0	0	0
21	1	0	0	0	0	0	0

Table 6.6 Estimation of number of animals per age in 2008

Age	Bearded Vulture	Chamois	Red deer female	Red deer male	Fallow deer	Roe deer	Sheep
1	0	988	978	254	125	1210	27776
2	0	987	780	192	110	1207	26944
3	0	890	625	154	103	1207	26135
4	0	889	500	124	95	1207	25352
5	0	889	400	99	89	1085	24591
6	0	795	240	60	83	1083	23854
7	0	792	195	48	77	1083	23137
8	6	690	155	38	71	959	22212
9	6	689	123	30	52	959	0
10	6	592	97	24	50	0	0
11	6	592	78	20	45	0	0
12	5	592	62	16	0	0	0
13	5	497	50	12	0	0	0
14	5	497	40	10	0	0	0
15	5	496	32	8	0	0	0
16	5	395	25	6	0	0	0
17	5	394	20	5	0	0	0
18	5	336	0	0	0	0	0
19	5	0	0	0	0	0	0
20	5	0	0	0	0	0	0
21	5	0	0	0	0	0	0

Table 6.7 Number of animals produced by the simulator

Year	Bearded Vulture	Pyrenean Chamois	Pyrenean Chamois	Red deer	Fallow deer	Roe Deer	Sheep
1994	20	9000		1000	600	1000	150000
1995	21	9541		1115	667	1213	152074
1996	21	10023		1263	710	1371	153951
1997	22	10590		1432	758	1568	156183
1998	23	11121		1617	808	1812	158571
1999	24	11718		1834	859	2106	161318
2000	25	12366		2087	908	2469	164391
2001	27	13032		2368	967	2906	167914
2002	28	13767		2705	1032	3459	171940
2003	29	14597		3067	1111	4132	174713
2004	31	15488	10000	3470	1202	4969	177973
2005	33	16468	10594	3917	1297	5883	181300
2006	35	17508	11133	4437	1399	6974	184790
2007	36	18647	11709	5004	1495	8272	188357
2008	**38**	**19866**	**12297**	**5631**	**1602**	**9774**	**192097**

MetaPlab: A Computational Framework for Metabolic P Systems

Alberto Castellini, Vincenzo Manca

Verona University, Computer Science Department,
Strada LeGrazie 15, 37134 Verona, Italy.
{alberto.castellini, vincenzo.manca}@univr.it

In this work the formalism of Metabolic P systems has been employed as a basis of a new computational plugin-based framework for modeling biological networks. This software architecture supports MP systems dynamics in a virtual laboratory, called MetaPlab. The Java implementation of the software is outlined and a specific plugin at work is described to highlight the internal functioning of the entire architecture.

1 Introduction

Systems biology copes with the quantitative analysis of biological systems by means of computational and mathematical models which assist biologists in developing experiments and testing hypothesis for complex systems understanding [12, 26]. On the other hand, new mathematical and computational techniques have been conceived to infer coherent theories and models from the huge amount of available data.

P systems were introduced by Gh. Păun in [23] as a new computational model inspired by the structure and functioning of the living cell. This approach is rooted in the context of formal language theory and it is essentially based on *multiset* rewriting and *membranes*. In the P systems theory many computational universality results have been achieved [24]. P systems seem especially apt to model biological systems, however their original mathematical setting was too abstract for expressing real biological phenomena.

Metabolic P systems, namely *MP systems*, are a class of P systems proved to be significant and successful for modeling biological phenomena related to metabolism (matter transformation, assimilation and expulsion in living organisms). They were conceived in [19] and subsequently extended in many works [4, 5, 15–18]. MP system dynamics is computed by a deterministic algorithm based on the *mass partition principle* which defines the transformation rate of object populations, according to a suitable generalization of chemical laws. This kind of rewriting-based and bio-inspired modeling overcomes some drawbacks of traditional Ordinary Differential Equations (ODE) allowing a new

insight about biological processes, which cannot be achieved by using the "glasses" of classical mathematics [2].

Equivalence results have been proved, in [9] and [7, 8], between MP systems and, respectively, autonomous ODE and Hybrid Functional Petri nets. The dynamics of several biological processes has been effectively modeled by means of MP systems, among them: the Belousov-Zhabotinsky reaction (in the Brusselator formulation) [4, 5], the Lotka-Volterra dynamics [4, 19], the SIR (Susceptible-Infected-Recovered) epidemic [4], the Protein Kinase C activation [5], the circadian rhythms, the mitotic cycles in early amphibian embryos [18], a Pseudomonas quorum sensing model [1, 6] and the *lac* operon gene regulatory mechanism in glycolytic pathway [7]. In order to simulate MP systems we developed a Java computational tool called *MPsim* [3]. The current release of the software, available at [10], is based on the theoretical framework described above, and it enables the graphical definition of MP models, their simulation and plotting of dynamics curves.

Recent work aims at deducing MP models, for given metabolic processes, from a suitable macroscopic observation of their behaviors along a certain number of steps. Indeed, the search of efficient and systematic methods to define MP systems from experimental data is a crucial point for their use in complex systems modeling. The solution of this *reverse-engineering* task is supported, into the MP systems framework, by the *Log-gain* theory [14, 15] which roots in allomeric principle [27]. The main result of this theory is the possibility of computing *reaction fluxes* at each step by solving a suitable linear equations system which combine stoichiometric information with other regulation constraints (by means of a sophisticated method for squaring and making univocally solvable the systems). This means that the knowledge of substances and parameters at each step provides the evaluation of reaction fluxes at that step. In this way, time-series of system states generate corresponding flux series, and from them, by standard regression techniques, the final regulation maps are deduced. This approach turned to be very effective in many cases and recently [20] it provided a model of a photosyntesis phenomenon, deduced by experimental time-series.

What seems to be peculiar of Log-gain theory is the strong connection with biological phenomena and its deep correlation with the allomeric principle, a typical concept of systems biology. Other general standard heuristics or evolutive techniques, already employed to estimate model structures and parameters [25], could be usefully combined with Log-gain method, in fact, the biological inspiration of this theory could add particular specificity to the wide spectrum potentiality of heuristics/evolutionary techniques by imposing constraints able to orientate the search of required solutions.

In this work, we propose a new plugin-based architecture that transforms the software MPsim from a simple simulator to a proper *virtual laboratory* which will be called *MetaPlab*. It assists biologists to understand internal mechanisms of biological systems and to forecast, in silico, their response to external stimuli, environmental condition alterations and structural changes. The Java implementation of MetaPlab ensures the

cross-platform portability of the software, which will be released under the GPL open-source license.

Several tools for modeling biological pathways are already available on-line. The most of them are based on ODE, such as *COPASI* [11], which enables to simulate biochemical networks and to estimate ODE parameters. It is a very powerful tool but its usage requires a deep knowledge of molecular kinetics, because the involved differential equations have an intrinsically microscopic nature. Petri nets have been employed by *Cell Illustrator*^{TM} [22], a software which graphically represents biological pathways by graphs and computes their temporal dynamics by a specific evolution algorithm [8]. Unfortunately, this tool can be used just to simulate biological behaviors, but it does not provide any support for the parameter estimation and the analysis of models. The new computational framework we propose in the following, instead, is based on an extensible set of plugins, namely Java tools for solving specific tasks relevant in the framework of MP systems, such as parameter estimation for regulative mechanisms of biological networks, simulation, visualization, graphical and statistical curve analysis, importation of biological networks from on-line databases, and possibly other aspects which would result to be relevant for further investigations.

In Section 2 we introduce some basic principles of MP systems and MP graphs, and we discuss a few biological problems which can be tackled by this modeling framework. Section 3 describes the new plugin-based architecture for a systematic management of these problems, and finally, a plugin for computing MP systems dynamics is presented in Section 4 with a complete description of its functioning.

2 MP systems: model and visualization

MP systems are deterministic P systems developed to model dynamics of biological phenomena related to metabolism. The notion of MP system we consider here generalizes the one given in [15, 18].

Definition 1. (MP system) *An MP system is a discrete dynamical system specified by a construct [14]:*

$$M = (X, R, V, Q, \Phi, \nu, \mu, \tau, q_0, \delta)$$

where X, R, V are finite sets of cardinality $n, m, k \in \mathbb{N}$ (the natural numbers) respectively.

1. $X = \{x_1, x_2, \ldots, x_n\}$ *is a set of* ***substances*** *(the types of molecules);*
2. $R = \{r_1, r_2, \ldots, r_m\}$ *is a set of* ***reactions*** *over X. A reaction r is represented in the arrow notation by a rewriting rule $\alpha_r \rightarrow \beta_r$ with α_r, β_r strings over X. The* ***stoichiometric matrix*** $\mathbb{A}$ *stores reactions stoichiometry, that is, $\mathbb{A} = (\mathbb{A}_{x,r} \mid x \in X, r \in R)$ where $\mathbb{A}_{x,r} = |\beta_r|_x - |\alpha_r|_x$, and $|\gamma|_x$ is the number of occurrences of the symbol x in the string γ;*

3. $V = \{v_1, v_2, \ldots, v_k\}$ *is a set of* ***parameters*** *(such as pressure, temperature, volume, pH, ...) equipped with a set* $\{h_v : \mathbb{N} \to \mathbb{R} \mid v \in V\}$ *of* ***parameter evolution functions****, where, for any* $i \in \mathbb{N}$*,* $h_v(i) \in \mathbb{R}$ *(the real numbers) is the value of parameter* v *at the step* i*;*
4. Q *is the set of* ***states****, seen as functions* $q : X \cup V \to \mathbb{R}$ *from substances and parameters to real numbers. A general state* q *can be identified as the vector* $q = (q(x_1), \ldots, q(x_n), q(v_1), \ldots, q(v_k))$ *of the values which* q *associates to the elements of* $X \cup V$*. We denote by* $q_{|X}$ *the restriction of* q *to the substances, and by* $q_{|V}$ *its restriction to the parameters;*
5. $\Phi = \{\varphi_r : Q \to \mathbb{R} \mid r \in R\}$ *is a set of* ***flux regulation maps****, where for any* $q \in Q$*,* $\varphi_r(q)$ *states the amount (moles) which is consumed/produced, in the state* q*, for every occurrence of a reactant/product of* r*. We define* $U(q) = (\varphi_r(q) \mid r \in R)$ *the* ***flux vector*** *at state* q*;*
6. ν *is a natural number which specifies the number of molecules of a (conventional) mole of* M*, as its* ***population unit****;*
7. μ *is a function which assigns to each* $x \in X$*, the* ***mass*** $\mu(x)$ *of a mole of* x *(with respect to some measure unit);*
8. τ *is the* ***temporal interval*** *between two consecutive observation steps;*
9. $q_0 \in Q$ *is the* ***initial state****;*
10. $\delta : \mathbb{N} \to Q$ *is the* ***dynamics*** *of the system. It can be identified as the vector* $\delta = (\delta(0), \delta(1), \delta(2), \ldots)$*, where* $\delta(0) = q_0$*, and* $\delta(i) = (\delta(i)_{|X}, \delta(i)_{|V})$ *is computed by the following autonomous first order difference equations:*

$$\delta(i+1)_{|X} = \mathbb{A} \times U(\delta(i)) + \delta(i)_{|X} \tag{28}$$

$$\delta(i+1)_{|V} = (h_v(i+1) \mid v \in V) \tag{29}$$

where $\mathbb{A}$ *is the stoichiometric matrix of* R *over* X*, of dimension* $n \times m$*, while* $\times$*,* $+$ *are the usual matrix product and vector sum. We introduce the symbol* $\delta_{<i}$ *to identify the finite vector* $(\delta(0), \delta(1), \ldots, \delta(i))$*.*

MP graphs, introduced in [18], are a natural representation of MP systems modeling biochemical reactions as bipartite graphs with two levels, in which the first level describes the *stoichiometry* of reactions, while the second level expresses the *regulation*, which tunes the flux of every reaction (i.e., the quantity of chemicals transformed at each step) depending on the state of the system (see for example Figure 2.1).

Given a metabolic process, some elements of a related MP system can be generally defined from a macroscopic observation of the system, while other elements should be computed by means of suitable mathematical techniques. For instance, if we deduce by experimental observations the set of substances (X, item 1 of Definition 1), the chemophysical parameters (V, item 3) and the reactions (R, item 2) involved in the biological process, and if we know the mathematical laws which regulate these reactions (Φ, item 5), then the system dynamics (δ, item 10) can be computed by the equations (28) and (29).

The *dynamics computation* task, just defined, is only one of several biologically inspired mathematical problems which can be tackled by MP systems. Table 2.1 collects a few of these tasks focusing on the known and the unknown elements of the related MP system. The second problem we propose is the *flux discovery*, which entails the computation of flux time-series $U(\delta(0)), \ldots, U(\delta(i-1))$ that yield an observed dynamics $\delta_{<i}$ of substances and parameters. A mathematical theory for solving this problem, called Log-gain theory, has been proposed in [14, 15], and some computational tools based on it are currently under construction. A third task is related to *regulation discovery*. It is a regression problem which aims at computing functions Φ which better interpolate a (known) flux time-series $U(\delta(0)), \ldots, U(\delta(i))$. They could be calculated by traditional regression methods [20] as well as by evolutionary computing techniques, such as genetic programming [13] and neural networks [21].

Problem	**Known elements**	**Unknown elements**
Dynamics computation	X, R, V, Φ, q_0	δ
Fluxes discovery	$X, R, V, U(\delta(0)), \delta_{<i}$	$U(\delta(1)), \ldots, U(\delta(i-1))$
Regulation discovery	$R, U(\delta(0)), \ldots, U(\delta(i)), \delta_{<i}$	Φ
Dynamics analysis	$X, R, V, \delta_{<i}$	Statistical params, etc.

Table 2.1 Some biologically inspired problems which can be tackled within the MP systems framework. Unknown elements should be computed from known elements by means of suitable mathematical techniques and computational tools.

The last problem listed in Table 2.1 concerns the *dynamics analysis*, a data-mining task which involves the discovery of new biological information from observed dynamics. It is related to the discovery of statistical parameters (e.g., dynamics and flux correlations), the clustering of observed time-series to detect the main actors of a biological system, and the analysis of the dynamical behaviors occurring from different (environmental and structural) conditions.

Of course, it could be very useful to systematically attack these and further bio-inspired problems by means of a set of computational tools suitably developed to satisfy biologists' needs. The software architecture proposed in the next section answers this request by supporting an extendable set of plugins, each dedicated to a specific task.

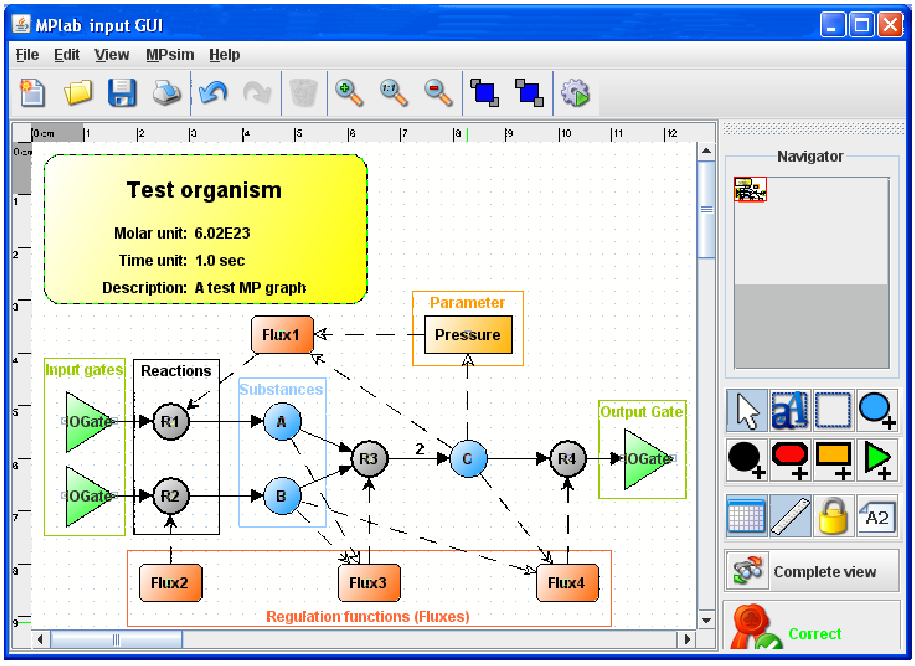


Fig. 2.1 An MP graph visualized by a graphical user interface of MetaPlab. Frame labels point out MP system elements in the MP graph representation. Substances, reactions and parameters describe the system stoichiometry, while fluxes express the system regulation.

3 A new plugin-based framework for processing MP systems

Here, we propose a computational structure which enables MetaPlab to systematically tackle the problems introduced in Table 2.1. Figure 3.2 depicts this framework, which involves four main layers: the first deals with the model definition and visualization by *MP graphs*, the second is dedicated to the representation and storing of MP systems by a suitable data structure called *MP store*, the third concerns with the processing of these data by means of computational units called *MP plugins*, and finally, the fourth arranges a set of *vistas* which support the MP systems analysis.

MetaPlab employs this framework to extend the functionalities and to improve the performances of MPsim 3, which only coped with the MP systems simulation. The new extendable data processing layer, described below, turns MetaPlab to be a proper "virtual laboratory" wherein MP plugins act as virtual tools for processing MP systems. In the following we show some implementation details of the new software architecture depicted in Figure 3.2. A technical description of the whole architecture will be published soon in the MetaPlab User Guide [28].

MP graphs. The leftmost layer of Figure 3.2 contains the MetaPlab input GUI, also

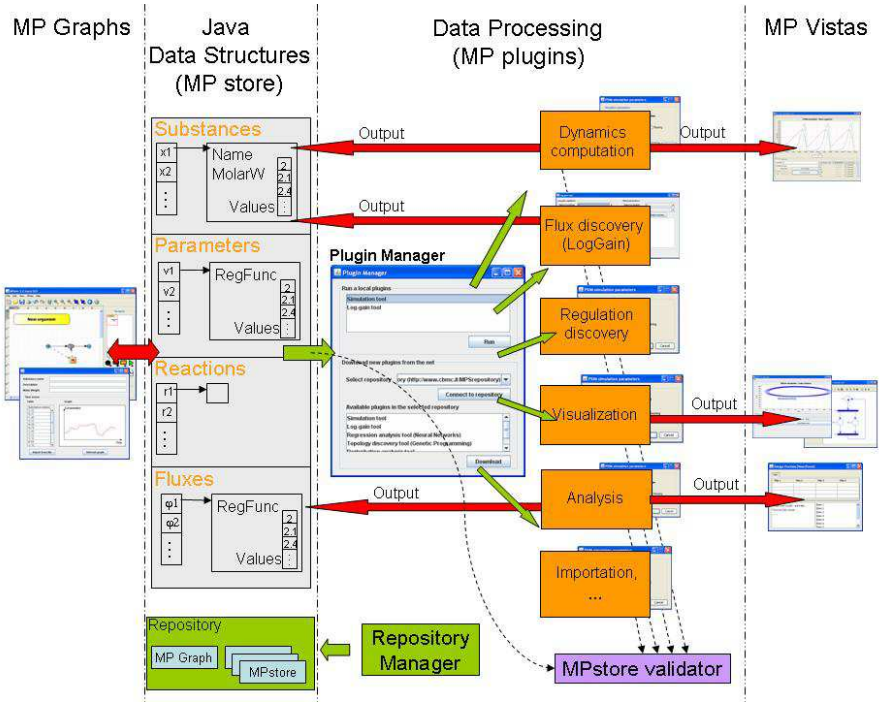


Fig. 3.2 The MetaPlab framework.

depicted in Figure 2.1. It is an easy-to-use graphical user interface which takes MP systems as inputs and visualizes them by means of MP graphs. The user drags MP graph elements from the right toolbar of Figure 2.1 to the central white panel. He or she specifies their internal parameters by filling in suitable fields, and connects the nodes by drawing arcs between them. Importation of observed time-series related to substances, parameters and fluxes dynamics is supported and a "network-oriented" visualization of MP system dynamics is provided by pop-up windows attached to each node. MP graphs loaded by this GUI are stored into MP store objects, which are defined below.

***MP store* data structure.** The second layer of Figure 3.2 consists of an object-oriented data structure called *MP store*. It has been designed to store all the elements of an MP system by suitable Java objects. Each *substance* $x \in X$ is mapped to an object which stores the substance name x, its molar weight $\mu(x)$ and the time-series of its dynamics $((\delta(i))(x) \mid i \in \mathbb{N})$. Each *parameter* $v \in V$ is implemented by an object having two main fields, the first stores the regulation function h_v as a string, while the second holds the time-series of its dynamics $((\delta(i))(v) \mid i \in \mathbb{N})$ as a vector of real numbers. *Flux* objects are very similar to parameters, in fact each flux stores the regulation function φ_r of a reaction r by a string field, and the related flux time-series $(\varphi_r(\delta(i)) \mid i \in \mathbb{N})$ by a vector. Finally, each *reaction* object implements a reaction rule $r \in R$ by a vector

of pointers that address the substances objects involved in r. Each pointer from a reaction r to a substance x has a related multiplicity field which stores the value $\mathbb{A}_{x,r}$ of the stoichiometric matrix. MP store is a crucial point of the new software architecture. Indeed, being the standard input of every plugin, it acts as a bridge between the *MP graph visualization* and the *data processing* layer described in the following.

Data processing. The third layer of Figure 3.2 represents the core of the new architecture, in fact, it concerns with a plugin-based module coping with the MP systems data processing. This layer is composed by *i)* an extendable set of Java *plugins*, listed on the right of the third layer, each equipped with specific input and (auxiliary) output GUIs, and *ii)* a *Plugin Manager*, depicted on the left of the third layer, which automatically loads MP plugins and makes them available to be launched. *MP plugins* are the MetaPlab processing units. Each of them is involved in a specific computational task, such as the dynamics computation of an MP system, the estimation of its regulation functions, the analysis of its dynamics, or the importation of metabolic pathways from databases. To accomplish one of these (or further) tasks, a plugin gets two possible **inputs**: an MP store object, which addresses the model visualized by the input GUI, and a set of auxiliary data, coming from a plugin-specific input GUI (if the plugin provides it). The plugin **outputs** may be saved into one or more MP store objects or they can be displayed by plugin-specific output GUIs (the *MP vistas* described below).

A plugin can be implemented by one or more Java classes, having methods for accomplishing some basic functions, such as, to return the plugin name and its description, to acquire the input, to perform the data processing, and to return the output. Further Java methods manage the plugin synchronization with the rest of the application. Once all the required methods have been implemented, the plugin is ready to be launched by means of MetaPlab. In this way, the compiled (.jar) file of the plugin should be placed into a specific folder, called *plugin directory*, in order to be automatically recognized and loaded by the Plugin Manager.

Figure 3.3 depicts the *Plugin Manager* GUI which enables the user to choose plugins from a list and to run them. The *upper side* of this window displays the available plugins. Each of them can be launched by selecting the related entry in the list and by clicking the underlying *Run* button. If a plugin saves its output as an MP store data structure, then further plugins can work on this output, getting it as an input. Whenever a plugin computation stops, the Plugin Manager is displayed, in order to give the user the chance to run another plugin. The *lower side* of the Plugin Manager is instead dedicated to deliver new plugins. In fact, due to the intrinsic open and easy structure, MP plugins can be implemented, following a few simple rules, by whoever wants to attack a specific modeling problem by MP systems. From this perspective, the forthcoming on-line repositories will enable the exchange of these computational tools among the MetaPlab users, thus encouraging their reuse. When an on-line repository is selected by the first text field, the subsequent text box automatically shows the list of plugins which can be downloaded from the repository. Soon, a web site dedicated to MetaPlab [28]

will support the download of new plugins and it will provide a complete documentation of the software.

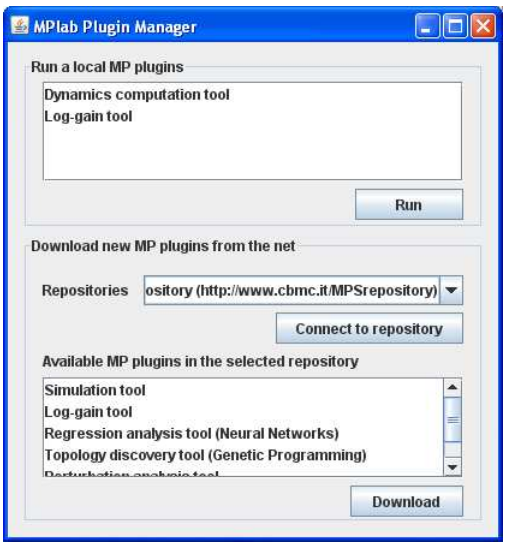


Fig. 3.3 The MetaPlab Plugin Manager. In the upper side the user can run plugins by choosing them from a list. The lower side allows the user to download new plugins from forthcoming on-line repositories.

MP vistas. The fourth level concerns with other ways of representing MP structures and MP dynamics, which can support the analysis of specific aspects of the modeling process. Some examples of these vistas are the jointly tracing of substances and parameters curves, the plotting of phase diagrams, and the visualization of statistical indexes.

Auxiliary modules. Two further modules are displayed in the bottom of Figure 3.2: the *MP store validator* and the *repository manager*. The first is a Java library which assists plugin designers to check the MP store consistency. The second manages the systematic storage and retrieval of the experiments related to a specific MP system.

4 A plugin for computing MP system dynamics

In this section we propose a plugin example to highlight the mechanisms underlying the plugin-based framework described above. The plugin we propose is a *simulator* which computes the dynamics of an MP system by applying the recurrent equations (28) and (29) of Definition 1. We remark that the plugin is simply a rearrangement of the stand-alone software MPsim 3. The main difference between the stand-alone simulator and the relative plugin version is that, the latter satisfies some structural requirements which

allow it to be automatically loaded by the Plugin Manager, to communicate with the MetaPlab input GUI and to exchange data with other plugins. All the details about these simple requirements will be published in the forthcoming MetaPlab Developer Guide [28].

Plugin functioning. At the beginning of the modeling process, we define an MP system by dragging substance, parameter and reaction nodes from the right toolbar of the input GUI (Figure 2.1). Then, we draw stoichiometric arcs, and we state both regulation functions and initial conditions. For example, let us imagine to define an MP graph for a typical metabolic process, as the mitotic oscillator already simulated by the stand-alone tool in [3].

After this input stage we open the Plugin Manager (Figure 3.3) which lists all the available plugins. We select the *dynamics computation tool* by choosing the related entry from the upper list, and we click the *Run* button, in order to start the plugin. The graphical user interface depicted on the left side of Figure 4.4 appears on the screen. By this window we state the number of steps to perform and then, we launch the dynamics computation process by the start button. When the process finishes, the substance, parameter and flux time-series, computed by the plugin, are automatically saved by an MP store which updates the MP graph displayed by the input GUI. Furthermore, the dynamics is plotted by the plugin output interface, depicted on the right of Figure 4.4, which shows the typical mitotic oscillations.

We finally remark that the just computed dynamics can be processed again by further plugins, as in a pipeline, because MP store objects preserve the format compatibility among all these tools. From this perspective MetaPlab widely increases the computational power of MPsim.

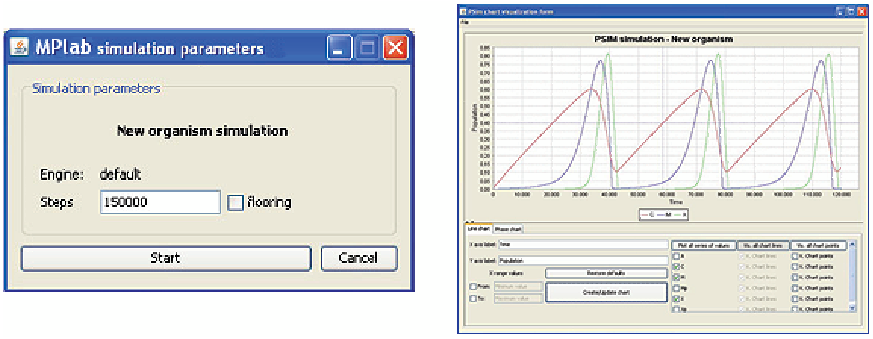


Fig. 4.4 On the left: The input graphical user interface of the dynamics computation plugin. **On the right:** the output graphical user interface of the dynamics computation plugin.

5 Conclusions and future works

This work has shown that several problems related to the modeling of biological networks can be systematically tackled by means of a set of computational tools integrated in a virtual laboratory, based on the MP systems theory.

The power of this laboratory tightly depends on the flexibility of the plugins architecture and will increase as much as we collect new plugins enriching the basic functionalities of our system. For example, at present we are almost ready to add a new plugin, based on the Log-gain theory [14], which compute the fluxes of a given MP system, deduced by a temporal series of observed states. We plan also to develop other plugins based on traditional regression techniques, neural networks and genetic programming, for obtaining flux maps from fluxes.

Other important functionalities of our virtual laboratory will be topics of further extensions. In particular we want to mention: *i)* plugins which compute suitable statistic coefficients and analyze the system stability when biological parameters change, *ii)* plugins dedicated to the network importation from the main on-line databases (by the SBML standard), *iii)* plugins able to map MP systems to other formalisms, such as ODE or Petri Nets, and to visualize MP models by means of alternative vistas.

Bibliography

[1] F. Bernardini, M. Gheorghe, and N. Krasnogor. Quorum sensing P systems. *Theoretical Computer Science*, 371(1-2):20–33, 2007.

[2] F. Bernardini and V. Manca. Dynamical aspects of P systems. *BioSystems*, 70:85–93, 2003.

[3] L. Bianco and A. Castellini. Psim: a computational platform for Metabolic P systems. In *LNCS 4860*, pages 1–20. Springer, 2007.

[4] L. Bianco, F. Fontana, G. Franco, and V. Manca. P systems for biological dynamics. In G. Ciobanu, G. Păun, and M. J. Pérez-Jiménez, editors, *Applications of Membrane Computing*, Natural Computing Series, pages 81–126. Springer, Berlin, 2006.

[5] L. Bianco, F. Fontana, and V. Manca. P systems with reaction maps. *International Journal of Foundations of Computer Science*, 17(1):27–48, 2006.

[6] L. Bianco, D. Pescini, P. Siepmann, N. Krasnogor, F. J. Romero-Campero, and M. Gheorghe. Towards a P systems pseudomonas quorum sensing model. In *LNCS 4361*, pages 197–214. Springer, 2006.

[7] A. Castellini, G. Franco, and V. Manca. Toward a representation of Hybrid Functional Petri Nets by MP systems. In *Proceedings of the 2nd International Workshop on Natural Computing, IWNC 2007, Nagoya University, Japan*. Springer Verlag Tokyo. To appear.

[8] A. Castellini, G. Franco, and V. Manca. Hybrid functional petri nets as MP systems. 2008. Submitted.

[9] F. Fontana and V. Manca. Discrete solutions to differential equations by metabolic P systems. *Theoretical Computer Science*, 372(2-3):165–182, 2007.
[10] Center for BioMedical Computing web site. Url: http://www.cbmc.it.
[11] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. COPASI a COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067–3074, 2006.
[12] H. Kitano. Computational systems biology. *Nature*, 420, 2002.
[13] J.R. Koza, M.A. Keane, M.J. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *Genetic Programming IV : Routine Human-Competitive Machine Intelligence (Genetic Programming)*. Springer, 2003.
[14] V. Manca. Log-gain principles for Metabolic P systems. In *G. Rozenberg Festschrift*. To appear.
[15] V. Manca. The Metabolic Algorithm: Principles and applications. *Theoretical Computer Science*. http://dx.doi.org/10.1016/j.tcs.2008.04.015. In print.
[16] V. Manca. Metabolic P systems for biochemical dynamics. *Progress in Natural Sciences*, 17(4):384–391, 2007.
[17] V. Manca. Discrete simulations of biochemical dynamics. In *LNCS 4848*, pages 231–235. Springer, 2008.
[18] V. Manca and L. Bianco. Biological networks in metabolic P systems. *BioSystems*, 91(3):489–498, 2008.
[19] V. Manca, L. Bianco, and F. Fontana. Evolutions and oscillations of P systems: Applications to biochemical phenomena. In *LNCS 3365*, pages 63–84. Springer, 2005.
[20] V. Manca, R. Pagliarini, and S. Zorzan. Toward an MP model of Non Photochemical Quenching. In *Pre-Proceeding of the 9-th Workshop on Membrane Computing*, 2008.
[21] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors. *Machine learning, neural and statistical classification*. Ellis Horwood, Upper Saddle River, NJ, USA, 1994.
[22] M. Nagasaki, A. Doi, H. Matsuno, and S. Miyano. Genomic object net: I. A platform for modelling and simulating biopathways. *Applied Bioinformatics*, 2(3):181–184, 2004.
[23] G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
[24] G. Păun. *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
[25] F. J. Romero-Campero, H. Cao, M. Camera, and N. Krasnogor. Structure and parameter estimation for cell systems biology models. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2008*. ACM Publisher, 2008. To appear.
[26] E. Voit, A. R. Neves, and H. Santos. The intricate side of systems biology. *PNAS*, 103(25):9452–9457, June 20 2006.
[27] L. von Bertalanffy. *General systems theory: foundations, developments, applications*. George Braziller Inc., New York, NY, 1967.
[28] MetaPlab website. Url: http://mplab.sci.univr.it.

Simulation of Membrane Computing for Secure Mobile Ad Hoc Networks

Das, Digendra K.

New York Nano-Bio-Molecular Information Technology (NYNBIT) Incubator,
Department of Mechanical and Industrial Engineering Technology
SUNY Institute of Technology, P.O. Box 3050, Utica, NY 13504-3050, U.S.A
das@sunyit.edu

This paper presents a bio inspired membrane computing simulation for an end-to-end, secure mobile ad hoc network (MANET). Mobile ad hoc nodes remain autonomous and organize themselves in an overall community of network devices to perform coordinated activities. Membrane computing or P systems, as distributed and parallel computational models, are developed as an abstract biological metaphor for mobile ad hoc network nodes. Mobile ad hoc membrane networks are modeled using a transitional P system simulation, with migration components and a guardian membrane that regulates interactions between the trusted network component and the unknown external environment.

1 Introduction

P systems belong to the class of theoretical natural computing models inspired from the way the live cells process chemical compounds, energy and information. They are highly parallel and based on the notion of a membrane structure [12]. Such a structure consists of several cell-like membranes recurrently placed inside a unique skin membrane. The regions delimited by a membrane structure are placed in multisets of objects, which evolve according to evolution rules associated with the regions. These objects placed in the regions delimited by the membranes, can be transformed into other objects and can pass through a membrane. The goal of this research was to investigate the feasibility of applying P system models as a framework to address an NP complete problem [10, 12], that of securing mobile ad hoc networks. Two basic classes of P systems, with symbol-objects and string-objects [8] were considered. Additionally P system models were developed based upon ambient mobile calculus for securing Mobile Ad hoc Networks [2, 15, 16].

A mobile ad hoc network (MANET) typically consists of a large number, potentially hundreds or thousands, of highly mobile data sources where users may be scattered over a wide area with little or no fixed network support. These networks must adapt rapidly to dynamic changes in network configurations. Mobile ad hoc networks consisting of

a wide variety of information sources and users require the use of distributed services and network protocols to solve the problems of mobility, weak signals and intermittent disconnection, dynamic reconfiguration, and limited power availability. The DasWebPS (P system) simulator was previously developed as part of the New York Nano-Bio-Molecular Information Technology (NYNBIT) incubator project. The DasWebPS simulator was used in the present research in order to model the transitional P systems that would replicate membrane agent behavior in a mobile ad hoc network. Results from the simulation were then abstracted using the Xholon [17] simulator's P system model for validation. The simulation was successfully completed in 10 iterations, terminating when the mobile ad hoc node exits the trusted environment (e.g. skin membrane).

2 Methodology

Transitional P systems and deterministic P systems with active membranes [13] have been simulated in various programming languages, and some of them have been used to solve NP-complete problems as Hamiltonian Path Problem (HPP), SAT, Knapsack, and partition problems. P systems with active membranes, input membrane and external output are simulated in CLIPS, and used to solve NP complete problems [14]. A more complex simulator written in Visual C++ for P systems with active membranes and catalytic P systems is presented in [3]. It provides a graphical simulator, interactive definition, visualization of a defined membrane system, a scalable graphical representation of the computation, and step-by-step observations of the membrane system behavior.

The DasWeb PS simulator allows for enhanced application level simulation and offers a user-friendly interface to the user. In addition it allows for debugging and visualization features and thus is a flexible P system development tool. In the DasWeb PS simulator all membrane computing applications are tree structures which become visible in the GUI once an application has been opened. The application tree contains three sub trees; Controller, View, and Model. At any time while the program is running, a single click on the parent Application node will display at the bottom of the GUI the name of the open model and the current time step.

A comparative analysis was undertaken inspired by [15, 16] applying an expressive ambient calculus to Membrane Computing, based on a similar structure and common concepts as a natural extension to the research described in [8,9]. The aim was to classify mobile network components according to their behavior, and at empowering sites with control capabilities which allow them to deny access to those agents whose behavior does not conform to the site's policy as described in [5]. Every site of a system conforms to: $k[M| > P]$ that consists of as an entity named k and structured in two layers: a communications agent P, possibly accessing local resources offered by the site, and a guardian membrane M, which regulates the interactions between the communications agent and the external environment. An agent P wishing to enter a site l must be verified by the guardian membrane before it is given a chance to enter site l. If the preliminary check succeeds, the agent is allowed to execute, otherwise it is rejected. In other words,

a membrane implements the policy each site wishes to enforce locally, by ruling on the requests of access of the incoming agents. This can be expressed by a migration rule of the form [6, page 2]:

$$k[M^k| > gol.P|Q]||l[M^l| > R] \rightarrow k[M^k| > Q]||l[M^l| > P|R] \text{ if } M^l \vdash^k P$$

The relevant parts here are P, the agent wishing to migrate from the original environment, *Group A* consisting of ambients (here k is the originating site and l being the receiving site), needs to be satisfied that P's behavior complies with its local policy. The latter is expressed by l's membrane, M^l. The judgment $M^l \vdash^k P$ represents l inspecting the incoming agent to verify that it upholds M^l.

Observe that in the formulation above $M^l \vdash^k P$ represents a runtime check of all incoming agents. Because of the fundamental assumption of openness, such kind of checks, undesirable as they might be, cannot be avoided. In order to reduce their impact on systems performance, and to make the runtime simulation as efficient as possible, it was necessary to adopt a strategy which allows for efficient agent verification. The resulting elementary notion of trust operates from the point of view that each l, the set of sites, is consistently partitioned between "trustworthy," "untrustworthy," and "unknown" sites. Then, in a situation like the one depicted in the rule above, it was assumed that l will be willing to accept a k-certified digest T of P's behavior from a trusted site k. It is therefore necessary to modify the primitive rule and the judgment $\vdash \mathbf{k}$ as in the refined migration rule [6, page 3]

$$k[M^k| > go_T l.P|Q]||l[M^l| > R] \rightarrow k[M^k| > Q]||l[M^l| > P|R] \text{ if } M^l \vdash^k_T P$$

As discussed in [6] the difference is expressed in $M^l \vdash^k_T P$. Here, l verifies the entire node P against Ml only if it does not trust k, the signer of P's certificate T, otherwise, it suffices for l to match Ml against the digest T carried by the primitive "go" together with P from k, so effectively shifting work from l to the originator of P.

3 Analysis

The novel contribution discussed in this paper is the notion of a "guardian membrane" used to implement and enforce different types of security related policies [6]. Here the focus was on the membrane agents' migration from site x to site y: the main operational mechanism is "between cells", rather than intra-site (i.e. local or intra-cellular) communication. Using these basic operations it was possible to develop a working model and subsequently simulate a transitional Membrane Computing model to verify the notion of policy enforced using membranes. This required a simple policy which only lists allowed actions and then proceeded to count action occurrences and then to apply nondeterministic Membrane Computing policies. Policies are the enforcement of rules concerning the behavior of single agents, and do not take into account "coalitional" behaviors. Here incoming agents, assumed to be benign, join clusters of resident agents

in order to perform cooperatively potentially harmful actions, or at least overrule the host site's policy. Those policies intended to be applied to the joint, composite behavior of the agents contained at a site are referred to as local or resident. Resident policies were explored as the final application of policy. In all the cases, the simulation adapts smoothly; one only needs to refine the information stored in the guardian membrane and their respective inspection mechanisms.

4 Discussion

The results reported in the analysis section reinforces the findings as reported in [1, 2] of an inherent security feature in ambient calculus, namely that the access to an ambient is authorized using the correct ambient name. The ambient name serves as a type of password for accessing the ambient. In the P system model considered above, the membranes from the Group A have the same names as the ambients (eg. nodes) they are guarding, in this way, when an agent is going to access some node, for example, the node named n, the corresponding membrane from the group P in the P system sends some multiset of objects to the membrane with name n. If the ambient with name n exists, that is the corresponding membrane with the name n, then with respect to the definition of transitional P systems operating within a dynamic network of membranes, the multiset of objects will be delivered to the membrane with address n. Otherwise, the action will be denied access until a membrane with name n appears.

There are several key components necessary to allow for secure mobile networking as described in [7]. First and foremost it requires a rich policy language, expressive enough to specify both authorization and information-flow policies as described in [4]. Together, these policies regulate the use and propagation of information throughout the network. During development of the Mobile Ad hoc membrane computing model, it was necessary to express constraints on allowable mobile node behaviors that can be checked statically. Policies appearing in node certificates are verified by hosts to ensure compliance with their own local data policies. Finally, during execution of the simulation, the policy language provides the vocabulary for authorization checks that are performed at run time to enforce access control. The Mobile Ad hoc model is based on an open system and enforces policy rules for specification of information-flow and authorization policies. The secure membrane networking architecture provides features for describing the locality of data sources and the security policies that govern them. This information is expressed as a collection of P system rules. The secure membrane networking architecture provides three services. First, it checks the certificate accompanying a node if it is asked to execute. This protects the host against malicious or corrupted code by ruling out potential flaws (e.g., buffer overflows, etc.) and ensures that the node complies with the host's local information-flow policy. This part of the security enforcement occurs before the node is allowed to enter the network. The certificate verifier is part of the trusted networking base. Second, the membrane networking system manages digital certificates that represent proof witnesses for the authorization and checks the node

to make certain it has been authenticated. And third, the membrane networking system provides secure inter-host communication. When, during the course of execution, a program needs to exchange data with another node in the network, it does so through the guardian membrane, which applies appropriate authentication and encryption to ensure that the underlying communication channel is secure.

5 Conclusion

Mobile ad hoc networking is increasingly characterized by the global scale of applications and the ubiquity of interactions between mobile components. Among the main features of the mobile ad hoc networking include secure information dissemination and location awareness, whereby nodes located at specific sites acts appropriately to local parameters and circumstances, that is, they exhibit "context-aware"; mobility, whereby information is dispatched from site to site to increase flexibility and expressivity; openness, reflecting the nature of global networks and embodying the permeating hypothesis of localized, partial knowledge of the networking environment. Such systems present enormous difficulties, both technical and conceptual, and are currently more at the stage of future prospects than that of conventional networking practice. Two concerns, however, appear to clearly be of a far reaching importance: security and mobility control, arising respectively from openness and from mobile ad hoc node and resource migrations.

Bibliography

[1] C. Braghin, D. Gorla, and V. Sassone. *A distributed calculus for role-based access control*. In Proc. of 17th Computer Security Foundations Workshop (CSFW'04), 48-60. IEEE Computer Society, 2004.

[2] L. Cardelli, G. Ghelli, and A. D. Gordon. *Ambient groups and mobility Membranes*. In International Conference IFIP TCS, number 1872 in Lecture Notes in Computer Science, pages 333-347. Springer, August 2000.

[3] G. Ciobanu, D. Paraschiv: *P system software simulator*. Fundamenta Informaticae,49, 1-3 (2002), 61-66.

[4] A. Cortesi, and R. Focardi. *Information Flow Security in Mobile Ambients*. In Proc. of International Workshop on Concurrency and Coordination CONCOORD'01, Lipari Island, July 2001, volume 54 of Electronic Notes in Theoretical Computer Science, Elsevier, 2001

[5] D. Gorla, M. Hennessy, V. Sassone, *Security Policies as Membranes in Systems for Global Computing*, FGUC 2004 Preliminary Version (2004). Final version published in Electronic Notes in Theoretical Computer Science.

[6] D. Gorla, M. Hennessy, and V. Sassone. *Security policies as membranes in systems for global computing*. In Proc. Workshop on Foundations of Global Ubiquitous Computing (FGUC), Preliminary version. London, ENTCS. Springer, 2004.

[7] L. Hash, P. Fitzgibbons, and D. Das, *Secure Architecture for Extensible Mobile Internet Transport Services: Implementation*, SPIE Defense and Security Symposium 2006, Session 2, April 17, 2006, Orlando, FL.

[8] S.N. Krishna, R. Rama: *A Variant of P Systems with Active Membranes: Solving NP-Complete Problems.* Romanian Journal of Information Science and Technology, 2, 4, 1999, 357-367.

[9] S.N. Krishna, R. Rama: *P Systems with Replicated Rewriting. Journal of Automata Languages, and Combinatorics*, 6, 3 (00), 345-350.

[10] A. Obtulowicz: *Deterministic P Systems for Solving SAT Problem*. Romanian Journal of Information Science and Technology. 4, 1-2(2001), 195-202.

[11] A. Paun: *On P Systems with Membrane Division.* In Unconventional Models of Computation (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer, London, 2000, 187-201.

[12] G. Paun. *P Systems with Active Membranes: Attacking NP-Complete Problems.* Journal of Automata, Languages and Combinatorics, 6, 1 (2001), 75-90.

[13] G. Paun. *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.

[14] M.J. Perez-Jimenez, F.J. Romero-Campero: *A CLIPS simulator for recognizer P systems with active membranes*, Proceedings 2nd Brainstorming Week on Membrane Computing, University of Sevilla Tech. Rep 01/2004, 387-413.

[15] I. Petre, L. Petre, *Mobile Ambients and P-Systems*, Journal of Universal Computer Science, vol. 5, no. 9 (1999), 588-598. Springer Publishing Company.

[16] V. Rogozhin, E. Boian, *Simulation of Mobile Ambients by P Systems. Part 2* in Proceedings of WMC 2003, Lecture Notes in Computer Science, Springer Publishing, Berlin, Germany, 2003, 431-442.

[17] Xholon Project, retrieved from: `http://www.primordion.com/Xholon/`

Usefulness States in New P System Communications Architectures

Juan Alberto de Frutos, Fernando Arroyo, Alberto Arteta

Universidad Politécnica de Madrid, Escuela Unversitaria de Informática,
Dpto. de Lenguajes, Proyectos y Sistemas Informáticos,
Crta. de Valencia Km. 7, 28031 Madrid, Spain
{jafrutos, farroyo, aarteta}@eui.upm.es

Dealing with distributed implementations of P systems, the bottleneck communication problem has arisen. When the number of membranes grows up, the network get congested. In agreement with this, several published works have presented an analysis for different architectures, which implement P systems in a distributed cluster of processors, allocating several membranes in the same processor. The purpose of these architectures is to reach a compromise between the massively parallel character of the system and the needed evolution step time to transit from one configuration of the system to the next one, solving the bottleneck communication problem.
The work presented here carries out an analysis of the semantics of the P systems, in several distributed architectures. It will be solved how to restructure P systems when dissolutions or inhibitions take place in membranes. Moreover, it will be also determined the extra information necessary at every communication step in order to allow all objects to arrive at their targets without penalizing the communication cost. This analysis will be performed on the base of usefulness states, which were presented in a previous work. The usefulness states allow each membrane of the system to know the set of membranes with which communication is possible at any time.

1 Introduction

Membrane Computing was introduced by Gh. Păun in [6], as a new branch of natural computing, inspired on living cells. Membrane systems establish a formal framework in which a simplified model of cells constitutes a computational device. Starting from a basic model, Transition P systems, many different variants have been considered; and many of them have been demostrated to be, in computational power, equivalent to the Turing Machine. Strictly talking from an implementational point of view and considering only the simplest model (Transition P systems), there are several challenges for researchers in order to get real implementatios of such systems. Today, one of the most interesting is to solve the communications bottleneck problem when the number of membranes grows up in the system. Accordingly with this fact, several works [8], [2] and [3]

present an analysis for distributed architectures based on allocating several membranes in the same processor, in order to reduce the number of external communications. These architectures allow certain degree of parallelism in application rules phase, as well as in the communication phase in a transition step during P System execution.

On the other hand, usefulness states were defined in [5] with two main goals. First and foremost, a usefulness state in a membrane represents the set of membranes to which objects can be sent by rules in the current evolution step. This information is essential to carry out a transition correctly. And second, usefulness states are used to improve the first phase -evolution rules application inside membranes- getting useful rules in a faster way. In [8], [2] and [3] the total time for an evolution step is computed, and what is more important is the fact that reducing the application phase time, the system obtains an important gain in the evolution step total time.

The goal of this paper is to fit usefulness states into communications architectures presented in [8], [2] and [3], solving the problem of membrane dissolution and membrane inhibition, not considered in those works. Furthermore, it will also considered the required information for objects to reach their respective target membranes. This information is based on the usefulness state concept.

2 Related works

At this point, several distributed architectures for implementing Transition P systems are described, and also the usefulness state concept is reviewed.

2.1 Communication architectures In order to face the communication problem in P System implementations, Tejedor et al. present in [8] an architecture named ”partially parallel evolution with partially parallel communication”. This architecture is based on the following pillars:

1. Membrane distribution. Several membranes are placed at each processor which will evolve, at worst, sequentially. Then, there are two kinds of communications:
   - Internal communications between membranes allocated at the same processor, with negligible communication time due to the use of shared memory techniques.
   - External communications between membranes placed in different processors
2. Proxies used to communicate processors. When a membrane wants to communicate with another one allocated at a different processor, uses a proxy. Therefore, external communications are carried out between proxies, no between membranes. This implies that each processor has a proxy which gathers objects from all membranes allocated on it, and after that it communicates with suitable proxies.

3. Tree topology of processors in order to minimize the total number of external communications in the system. Proxies only communicate with their parent and children proxies. Figure 2.1 shows an example of a membrane structure for a transition P System and its distribution in an architecture with four processors.

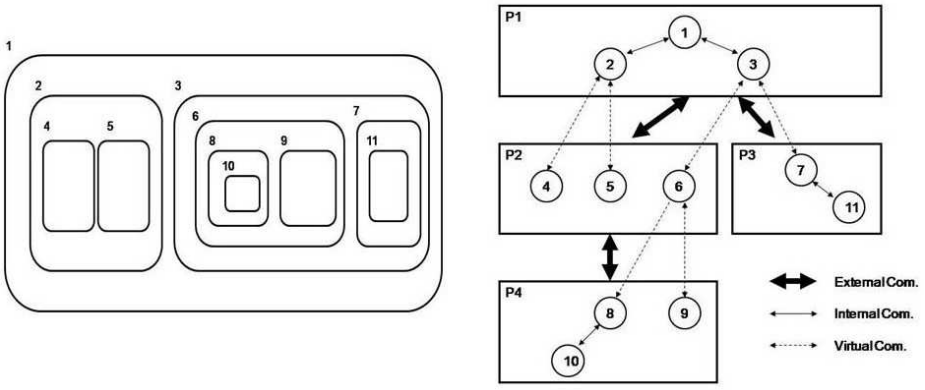


Fig. 2.1 Membrane distribution in processors

4. Token passing in the communications to prevent from collisions and network congestion. A communication order is established through a token, and then only one proxy tries to communicate at any moment. This token travels through a depth search sequence in the topology of processors tree. In the architecture of figure 2.1, the order in communications would be the following: P1 to P2, P2 to P4, P4 to P2, P2 to P1, P1 to P3 and finally P3 to P1.

More recently, Bravo et al. [2] have proposed a variant of this architecture. Membranes are placed in slave processors and a new processor is introduced acting as master. Slaves apply rules and send to the master multisets of objects whose targets are in a different slave. Master processor redistributes the multisets to its own slaves. This architecture keeps the parallelization in the application phase obtained in [8], but also it seeks for parallelizing the rule application phase in some processors with the communication phase in others. This produces the reduction of the evolution step time in the system.

An evolution of the last architecture was proposed in [3] by Bravo et al. Now, several master processors in a hierarchical way are used. This fact allows the parallelization of external communication and drastically increases the parallelization of application rules and external communication phases. As result, a better evolution step time in the system is obtained.

2.2 Uselfulness states. The usefulness state concept at membranes of a P System is introduced in [5]. This state allows to any membrane to know the set of children

membranes to communicate with (membrane context). This information is necessary to determine the set of rules to be applied in a evolution step, and it changes dynamically when membranes are dissolved or inhibited in the P System.

The set of usefulness states for a membrane j in a Transition P System can be obtained statically, that is, at analysis time, as we can see in [5]. One usefulness state in a membrane represents a valid context for that membrane, that is, a context that can be reached after an evolution step. As membrane context can change dynamically, transitions among states are also defined in [5].

From a given usefulness state can be obtained the set of useful rules. A rule is useful in an evolution step if all its targets are adjacent, not dissolved and not inhibited, then communication is feasible.

Figure 2.2 represents our example of P System. In this case, only rules associated to membrane 3 are detailed. Symbol δ in membranes 6, 9, 10 and 11 represents the possibility of these membranes to be dissolved by the application of some rules inside them. The symbol τ represents the possibility of inhibiting for membranes 6, 7 and 11 by the same cause. Usefulness states for membrane 3 are depicted in table 2.1, together with their contexts and useful rules associated.

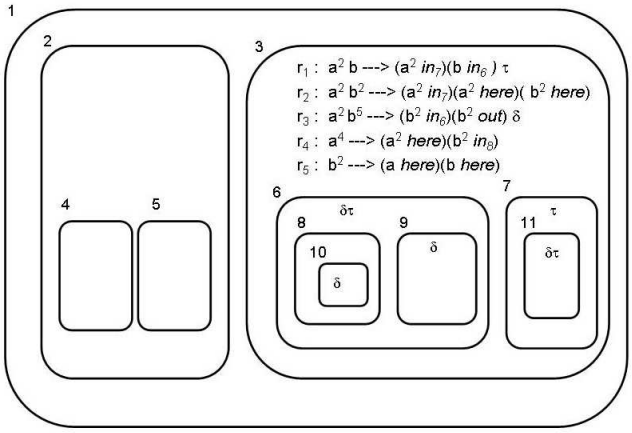


Fig. 2.2 Dissolving and inhibiting capabilities in membranes

Tables defining transition among states are also defined at analysis time. Suitable transitions take place when a child membrane of the current context changes its permeability. In such a way that, during system execution, membranes will obtain the set of useful evolution rules directly from their usefulness states, without any computation.

From an implementational point of view, problems arise when membranes have a high

Usefulness State	**Context**	**Useful rules**
q_0	$\{6, 7\}$	r_1, r_2, r_3, r_5
q_1	$\{6\}$	r_3, r_5
q_2	$\{8, 9, 7\}$	r_2, r_4, r_5
q_3	$\{8, 9\}$	r_4, r_5
q_4	$\{8, 7\}$	r_2, r_4, r_5
q_5	$\{9\}$	r_5
q_6	$\{7\}$	r_2, r_5
q_7	$\emptyset$	r_5

Table 2.1 Usefulness states for membrane 3

number of states, which cause transition tables to grow up. That is why in [5] it is proposed to encode usefulness states in order to avoid transition tables. Each one of usefulness states is encoded depending on its context, hence transitions are carried out directly in the code. Accordingly with this idea, two definitions are introduced:

Total Context for membrane j. It is the set made up of all membranes that eventually can become children of membrane j. Therefore, all contexts are included in the total context.

$$TC(j) = Child\_Of(j) \bigcup_{j_k \in Child\_D(j)} TC(j_k) \tag{30}$$

where $Child\_Of(j)$ is the set of all membrane j children in the initial membrane structure; and where $Child\_D(j)$ is the set of membrane j children that can be dissolved.

Normalized Total Context for membrane j. It is defined as the $TC(j)$ sorted in depth and in pre-order.

$$TC_{Normal}(j) = (j_1, TC_{Normal}(j_1), \ldots, j_n, TC_{Normal}(j_n)) \tag{31}$$

where $j_k \in Child\_Of(j)$ from left to right in μ, that is, in the initial membrane structure; and $TC_{Normal}(j_k)$ is considered as null if membrane j_k has not dissolving capability. For instance, in our P system, $TC_{Normal}(3) = \{6, 8, 9, 7\}$.

Each one of the usefulness states of a membrane j is encoded by $TC_{Normal}(j)$ depending on its context, with binary logic. The value 1 represents that the membrane belongs to the state context. For example, the usefulness state q_0 of membrane 3, representing the context $\{6, 7\}$, is encoded as 1001.

If $q^j(t) = (i_1, \ldots, i_k, \ldots, i_n)$ encoded by $TC_{Normal}(j)$ is the usefulness state for membrane j at time t, the transitional logic will be the following:

1. If membrane i_k at time t is inhibited, then: $q^j(t+1) = (i_1, \ldots, 0, \ldots, i_n)$
2. If membrane i_k at time t comes back to be permeable, then: $q^j(t+1) = (i_1, \ldots, 1, \ldots, i_n)$
3. If membrane i_k at time t is dissolved, it has to send its usefulness state $q^{i_j}(t)$, encoded by its normalized total context $TC_{Normal}(i_k)$, to the membrane j. Considering formula 31, the usefulness state for membrane j can be expressed in a deeper way as $q^j(t)$ = (i_1, ... ,i_k, $TC_{Normal}(i_k)$, ... ,i_n). Then, the transition obtained for membrane j is $q^j(t+1)$ = (i_1, ... ,0, $q^{i_j}(t)$, ... ,i_n)

In the proposed example, if membrane 3 is in the usefulness state $q^3(t) = 1001$, encoded by $TC_{Normal}(3) = \{6, 8, 9, 7\}$ and membrane 6 is dissolved in $q^6(t) = 11$ encoded by $TC_{Normal}(6) = \{8, 9\}$, it is obtained the transition $q^3(t+1) = 0111$

3 Usefulness states updating in membrane dissolution and inhibition

In order to fit properly usefulness states updating, we will previously describe the sucession of tasks that are carried out in an evolution step before communications initiate, that is, in the phase of rules application inside a membrane.

1. Active rules are obtained at every membrane of the system.
2. Active rules are applied in a maximal parallel and non deterministic way at every membrane of the system.
3. Each membrane of the system determines the result of rules application. The following information is obtained:

   - Objects which are produced and remain at the same membrane.
   - Objects which are produced and have as target an adjacent membrane of the P System. These objects will be sent to the proxy of the processor in which the membrane is placed. This proxy will be in charge of collecting objects and sending them to their respective targets, as will be described in section 5.
   - A new permeability state for the membrane, which is computed following the figure 3.3 automaton. This automaton represents transitions among membranes states based on the resulting dissolution and inhibition in the applied rules. The membrane will notify its new permeability state to the proxy only in case of changing.

When a proxy receives the information sent by a membrane -new permeability state and current usefulness state in case of dissolution- it is necessary to carry out the following tasks:

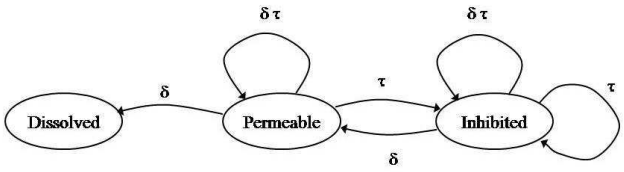


Fig. 3.3 Membrane permeability states

1. The proxy has to find out the father membrane. It is necessary to consider that it can change dynamically, as membranes are dissolved. Furthermore, the father may be allocated in another processor.
2. The proxy has to notify the new situation to the father. The latter will update its usefulness state according to this situation, as it has been shown in section 2.2

In order to achive these goals, the proxy must know the membrane structure, as regards membranes allocated in the proxy processor. And for each one of them, the proxy must know the following information:

- **j**: membrane identifier.
- **D(j)**: Dissolved. The value will be true if membrane j is dissolved.
- **TCL(j)**: Total Context Lenght. This value is computed in analysis time following the formula:

$$TCL(j) = \sum_{k=1}^{n} (1 \ + \ TCL'(j_k)) \tag{32}$$

where $j_k \in Child\_Of(j)$ and

$$TCL'(j_k) = \left\{ \begin{array}{ll} TCL(j_k) & \text{if } j_k \text{ has dissolving capability} \\ 0 & \text{otherwise} \end{array} \right\}$$

- **PFTC(j)**: Position at Father Total Context. This value is the membrane j position at the normalized father total context. This value is computed from the initial structure in analysis time following the formula:

$$PTCF(j_i) = 1 \ + \ \sum_{k=1}^{i-1} (1 \ + \ TCL'(j_k)) \tag{33}$$

where $j_k \ \in Child\_Of(j)$ at the left of j_i in μ

For instance, as $TC_{NORMAL}(3)$ = {6, 8, 9, 7}, values of $PFCT$ for both children membranes are obtained from this total context. Specifically, $PFTC(6)$ = 1 and $PFTC(7)$ = 4.

- **USM(j)**: Usefulness State Mask. The membrane j will make use of this mask in the usefulness state updating process.

Following with the example in figure 2.1, figure 3.4 represents the stored information in proxies. The values for $TLC(j)$ and $PFTC(j)$ are worked out from the corresponding normalized total context, which are obtained taking into account dissolving and inhibiting capabilities of membranes, depicted in figure 2.2.

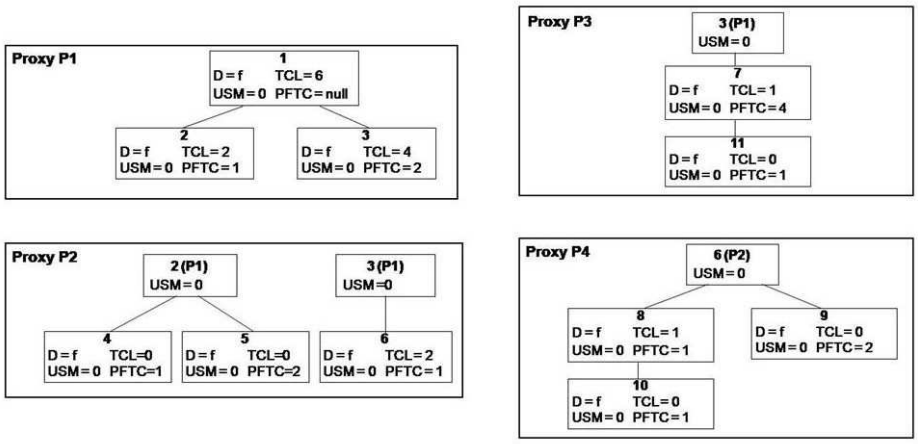


Fig. 3.4 Information stored in proxies to update usefulness states

The proxy looks for the father of the membrane which has changed the permeability state, going up in the membrane structure. In this case, it is necessary to use $D(j)$ to find a not dissolved membrane.

The proxy has also to prepare the suitable information for father membrane in order to update its usefulness state. The updating process is performed by changing the bit representing the membrane which has change its permeability state. This operation is performed through a XOR between the usefulness state and the $USM(j)$ field. As it is shown in 2.2, the following cases can be found:

- Inhibition of a child membrane. The position associated to the child membrane in the usefulness state has to be changed from 1 to 0, which means that communication is not possible for the next evolution step. A XOR operation with a bit 1 reaches this change. For instance, let us suppose that membrane 3 in our P System has the usefulness state 1001. As $TC_{Normal}(3) = \{6, 8, 9, 7\}$, this state represents the context {6,7}. Let us also suppose that membrane 7 is inhibited at this time. The usefulness state for membrane 3 is updated as follows:
  $USM(3)$ = 0001 (bit 1 for membrane 7)
  1001 XOR 0001 = 1000 (Context(3) = $\{6\}$)
- Removing inhibition of a child membrane. The position associated to this membrane has to be changed from 0 to 1. This represents that the child membrane accepts objects for the next evolution step. Again, a XOR operation with a bit 1

reaches the change.

- Dissolution of a child membrane. The bit representing the child membrane in the total context has to be changed from 1 to 0. Moreover, several of the following positions in the normalized total context represent the context of the dissolved membrane, as formula 31 shows, and necessarily these bits have to be replaced with the usefulness state of the dissolved membrane. One more time, these changes can be done with a XOR operation between the usefulness state and the mask stored in the $USM(j)$ field. For instance, let us suppose that the usefulness state of membrane 3 is 1001, representing context {6, 7}, and membrane 6 is dissolved in the usefulness state 10. As the $TC_{Normal}(6) = \{8, 9\}$, this state represents context $\{8\}$. The usefulness state of membrane 3 would be updated in the following way:
  $USM(3)$ = 1100 (bit 1 for 6, followed by its usefulness state)
  1001 XOR 1100 = 0101 (Context(3) = {8, 7})

The main problem now is to exactly determine the position of this information in the binary mask, that is, in the field $USM(j)$. In order to do this, it is necessary the $PFTC(j)$ field. The proxy goes up in the membrane structure looking for the father membrane, and simultaneously performing the addition of $PFTC(j)$ fields for every dissolved membranes found in the path.

As an example, let us suppose that membrane 9 is dissolved in a evolution step in which membrane 6 was already dissolved in a previous step, in such a way that membrane 3 is the membrane 9 father. The $USM(3)$ field can be obtained in the following way:

Information = 1 (membrane 9, followed by its usefulness state)

Position = $PFTC(9)$ + $PFTC(6)$ = 3

Lenght = $TCL(3)$ = 4

$USM(3)$ = 0010 (as $TC_{Normal}(3) = \{6, 8, 9, 7\} \Rightarrow$ 9 dissolution)

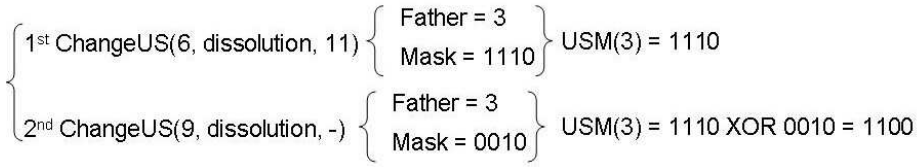
When a membrane j changes its permeability, the algorithm *ChangeUS* (Change Usefulness State, figure 3.5) will carry out this process. The operator + represents strings concatenation and 0^n represents a string with n symbols 0.

In an evolution step, it may happen that several children membranes change their permeabilities, involving the same father. Therefore, the usefulness state of a membrane has to be modified with several masks. No matter the order in which the proxy processes permeability changes, conmuntative and associative properties of XOR operation allow to obtain the value for $USM(j)$ correctly . Let us suppose, in our example, that membranes 6 and 9 are dissolved in the same evolution step. If proxy processes membrane 6

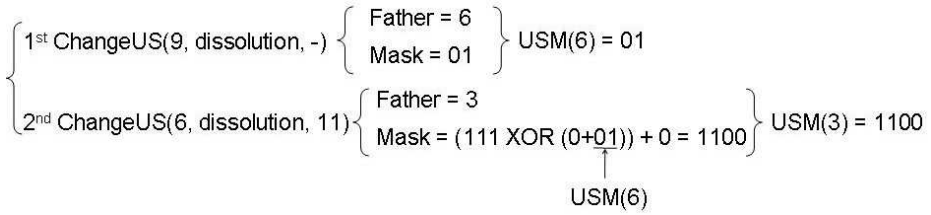
```
ChangeUS ( j, NewPerm, UState )
(1)   Mask <-- 1
(2)   IF NewPerm = Dissolved THEN BEGIN
(3)           Mask <-- Mask + UState
(4)           Mask <-- Mask XOR (0 + USM (j))
                    (* Send up current USM of membrane j *)
(5)     END
(6)   Mask <-- 0 PFTC(j)-1 + Mask
(7)   Target <-- Father ( j )
(8)   WHILE D(target) AND NOT OutProcessor(Target)
(9)     DO BEGIN
(10)          Mask <--  0 PFTC(Target) + Mask
(11)          Target <-- Father (Target)
(12)     END
(13)  IF NOT OutProcessor(Target) THEN
(14)          Mask <-- Mask + 0 TCL(Target) - | Mask |
                          (* Fit Mask in the Total Context *)
(15)  USM (Target) <-- USM (Target) XOR Mask
```

Fig. 3.5 Algorithm used to obtain the USM field for membrane j father

before, the resulting $USM(3)$ is procesed as follows:



Both dissolutions have been considered owing to XOR operation in line 15. On the other hand, if proxy processes membrane 9 before, the resulting $UMS(3)$ is procesed as follows:



When membrane 6 is dissolved $UMS(6)$ is inherited by the father membrane, that is $UMS(3)$, through XOR operation in line 4.

Finally, it is important to note that the father membrane may be placed in a different processor; therefore the process is carried out by several processors in a distributed way. The algorithm of figure 3.5 deals with this situation in lines 8 and 13, in which *OutProcessor* checks if the target membrane is allocated in other processor. Section 6 deals with communications in order to reach a distributed process.

4 Encoding targets of evolution rules within total context

Evolution rules in transition P systems have the form $u \rightarrow v$, $u \rightarrow v\ \delta$ or $u \rightarrow v\ \tau$, with $u \in O^+$ and $v \in (O^+ \times TAR)^*$, where O is the alphabet of objects, and $TAR = \{here, out\} \cup \{in_j \mid j \text{ is a membrane label}\}$. Symbol δ represents membrane dissolution, while symbol τ represents membrane inhibition.

Usually, transition P systems implementations up to now [4] [7] require to store a membrane identification for every target in_j in every rule in every membrane. In this paper a compact representation for evolution rules consequent based on the concept of total context is presented. It allows to represent targets without membranes identifications, what reduce significantly the necesary space to store rules. Moreover, this representation allows proxies to find any membrane target in a precise way.

The total context of a membrane is obtained at analysis time, and it encodes any possible in_j target for evolution rules of the membrane. Hence, adding a binary mask of length equal to membrane total context length, it is possible to control if a rule sends objects to a determined child membrane with label j. It is expressed setting to 1 the j position in the binary mask.

In addition, we propose four bits more in order to encode the complete consequent of a rule r_k, two for targets here (b_h^k) and out (b_o^k) respectively and two for representing membrane dissolution (b_δ^k) and inhibition (b_τ^k). Figure 4.6 shows the proposed encoding for a rule consequent. Besides the sequence of bits, each target has a multiset associated, represented as $M_h^k\ M_o^k\ M_1^k\ \ldots\ M_n^k$.

On the other hand, the antecedent of a rule r_k can be represented with another multiset: M_a^k.

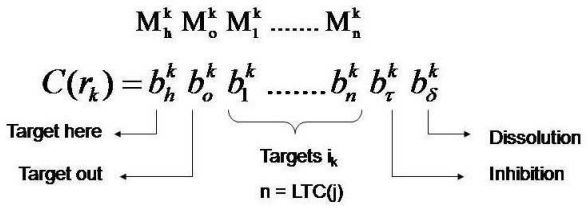


Fig. 4.6 Encoding for a rule consequent

Table 4.2 contains the encoded consequent of membrane 3 rules in our example. Let us remind that the normalized total context for this membrane is {6,8,9,7}

Rule	**Encoding**	**Multisets**
$r_1 : a^2 b \rightarrow (a^2\ in_7)(b\ in_6)\ \tau$	00100110	$M_1^1 = b,\ M_4^1 = a^2$
$r_2 : a^2 b^2 \rightarrow (a^2\ in_7)(a^2\ here)(b^2\ here)$	10000100	$M_h^2 = a^2 b^2,\ M_4^2 = a^2$
$r_3 : a^2 b^5 \rightarrow (b^2\ in_6)(b^2\ out)\ \delta$	01100001	$M_o^3 = b^2,\ M_1^3 = b^2$
$r_4 : a^4 \rightarrow (a^2\ here)(b\ in_8)$	10010000	$M_h^4 = a^2,\ M_2^4 = b$
$r_5 : b^2 \rightarrow (a\ here)(b\ here)$	10000000	$M_h^5 = ab$

Table 4.2 Encoding consequent of membrane 3 rules

In section 3 it was enumerated the task list to be performed in evolution rules application phase in membranes. Let us explain how can be used and computed the resulting evolution rule using this compact representation of binary mask and multiset of objects. Let $M_R(p) = r_1^{n_1} \ldots r_m^{n_m} = \sum_{i=1}^{m} n_i r_i$ be the multiset of rules to be applied in the evolution step p, where n_i means the number of times the rule r_i has to be applied. Then, it is needed to compute:

- $C(p)$, the sequence of bits, encoding targets, for the multiset of evolution rules consecuent ($b_h\ b_o\ b_1\ \ldots\ b_n$)

$$C(p) = OR_{\forall\, r_i \in M_R(p)}\, C(r_i) \tag{34}$$

- $M_h(p)$, $M_o(p)$, $M_1(p)$, ..., $M_n(p)$, the list of multisets of objects associated to $C(p)$.

$$M_h(p) = \sum_{\forall\, r_i \in M_R(p)} n_i\, M_h^i \tag{35}$$

$$M_o(p) = \sum_{\forall\, r_i \in M_R(p)} n_i\, M_o^i \tag{36}$$

$$M_j(p) = \sum_{\forall\, r_i \in M_R(p)} n_i\, M_j^i \tag{37}$$

- And finally, $M_a(p)$ the antecedent of the multiset of evolution rules.

$$M_a(p) = \sum_{\forall\, r_i \in M_R(p)} n_i\, M_a^i \tag{38}$$

When this process finishes, membranes proceed to data delivery:

- Multisets $M_h(p)$ and $M_a(p)$ will be applied directly to the membrane. Considering w the multiset of objects placed in the membrane at the beginning of the evolution step, w is updated by:

$$w = w - M_a(p) + M_h(p) \tag{39}$$

- $b_o\ b_1\ \ldots\ b_n$, together with $M_o(p)\ M_1(p)\ \ldots\ M_n(p)$, will be sent to the proxy processor.
- $b_\delta\ b_\tau$ will be used to find out changes of permeability, as the automaton in figure 3.3 shows. If b_δ is equal to 1, the transition δ is applied; otherwise if b_τ is equal to 1, the transition τ is applied; finally, the transition $\delta\ \tau$ is applied if both b_δ and b_τ are equal to 1. In the case of permeability change, membrane will notify the new permeability state to the proxy, in order to update the usefulness state of its father, as it is detailed in section 3.

5 Targets search in proxies

When a membrane has to send objects to its adjacent membranes, it uses the proxy. The membrane sends to the proxy a pair of data:

$$(Targets, MS)$$

Where $Targets$ is a binary sequence encoding labels of target membranes ($b_o b_1\ \ldots\ b_n$) and MS is the sequence of multisets associated to each one of the target membranes ($M_o(p)\ M_1(p)\ \ldots\ M_n(p)$). At this moment, the proxy has to perform the following tasks:

1. Target membrane for $M_o(p)$ is the father membrane. Hence the proxy will go up in the membrane structure until finding out the first not dissolved membrane.
2. Target membranes for $M_1(p)\ \ldots\ M_n(p)$ are encoded by $b_1\ \ldots\ b_n$. Hence, the proxy needs to analyse the Normalized Total Context of the source membrane. Considering equation (2) for Normalized Total Context of a given membrane, for every child membrane the proxy has to keep two important data: the dissolving capability of this membrane, and the length of its normalized total context.

As consequence, in order to perform targeting search, proxy has to store the following information related to membranes allocated on its processor:

- **D(j)**: Dissolved. The value will be true if membrane j is dissolved.
- **DC(j)**: Dissolving Capability. Its value will be true if there is any evolution rule which could dissolve the membrane j. It is obtained at analysis time.
- **TCL(j)**: Total Context Length.
- **M(j)**: Multiset for membrane j. When proxy determines that membrane j is a target, it stores temporally the suitable multiset in the $M(j)$ field.

Moreover, it is also necessary to note that one or more targets could be placed at different processors. Hence, the proxy has to prepare properly some information to send them, because search of targets must continue on these processors. So, proxy has to store some data about membranes placed in other processors with which there are established connection -virtual connections in figure 2.1-. The needed data for the proxy are:

- **DC(j)**
- **TCL(j)**
- **M(j)**
- **Targets(j)**: To store a sequence of bits encoding a list of targets.
- **MS(j)**: To store a list of multisets associated to the sequence of targets. This field and the previous one are needed only for membranes with dissolving capability.

Figure 5.7 shows the required information by proxies of the processors depicted in figure 2.2.

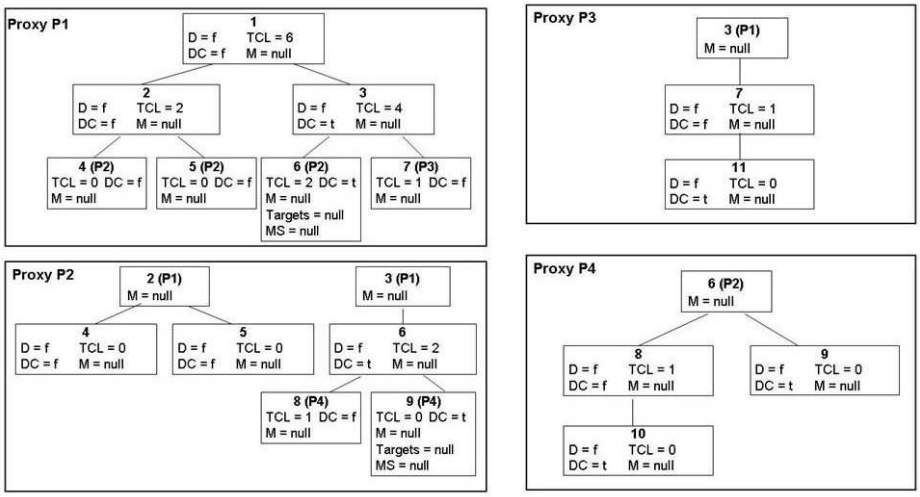


Fig. 5.7 Information stored in proxies to search targets

5.3 Target search for $M_o(p)$ The algorithm presented here (*Target_Out*) looks for the father of membrane j in order to send it the multiset $M_o(p)$. In line 5, $M_o(p)$ is assigned to the temporary field M of the father. Line 3 consider the situation in which search has to be continued in another processor, then the partial result remains in a field M awaiting to be sent to the appropiate processor. Section 6 of this paper deals with communications in architectures.

```
Target_Out ( j , b_o, M_o)
(1)   IF b_o = 1 THEN BEGIN
(2)        Target <-- Father ( j )
(3)        WHILE D(target) AND NOT OutProcessor(Target)
(4)             Target <-- Father (Target)
(5)        M (Target) <-- M_o
(6)   END
```

Fig. 5.8 Target search for multiset $M_o(p)$

5.4 Targets search for $M_1(p)$ to $M_n(p)$ The proxy has to interpret the normalized total context of the source membrane. With this aim, the proxy will go down into the sub-tree of the membrane structure, starting from the source membrane, in depth and in pre-order. When a membrane j has not dissolving capability ($DC(j)$) the analysis of this branch of the sub-tree is finished.

The recursive algorithm in figure 5.9 describes the search of targets from the source membrane j, the sequence of bits, encoding targets ($Targets$) and the list of multisets (MS) associated to targets. The algorithm visits children membranes from left to right. When the corresponding bit b_i is equal to 1, the multiset $M_i(p)$ is associated to the child membrane (line 7). Otherwise the child membrane is not a target, but if it has dissolving capability ($DC(j)$) then the search has to be continued from b_{i+1} into the normalized total context of the child membrane, as equation (2) shows. In case of the child membrane were allocated in the same processor, the search continues in the child membrane by making a recursive call in line 17. Otherwise, the information corresponding to the normalized total context of the child membrane is stored in $Targets(j)$ and $MS(j)$ fields in order to continue searching in the appropriate processor (lines 19 and 20)

An additional detail of the algorithm is the following: if b_i is equal to 1 and the child membrane has dissolving capability, it has to skip the total context of the current child membrane, because these membranes are not possible targets (line 8).

5.5 Membrane dissolution As it has been set before, the proxy has to execute algorithms *Target_Out* and *Target_In* for every membrane placed at the processor which require sending objects. Moreover, it has to execute the algorithm *ChangeUS* for every membrane which notifies a change of permeability.

Nevertheless, membrane dissolution has not been solved yet. Dissolution takes place when an evolution step finishes. Then, objects remaining inside the membrane have to pass to its father. In this way, when membrane j is going to be dissolved, we have to bear in mind the following:

```
Targets_In ( j , Targets, MS)
(1)   i <-- 1
(2)   FOR EACH k CHILD OF j FROM LEFT TO RIGHT BEGIN
(3)     IF Targets[i] = 1 THEN   (* b_i = 1 ==> membrane j is a target *)
(4)     BEGIN
(5)       IF NOT OutProcessor( k ) AND D( k )  (* k dissolved in current step *)
(6)         THEN Target_Out ( k, 1, M[ i ] )
(7)         ELSE M( k ) <-- M( k ) + MS[ i ]   (* j is the target for M_i *)
(8)       IF DC( k ) THEN i <-- i + TCL( k ) +1  (* Skip the total context *)
(9)                  ELSE i <-- i + 1
(10)    END
(11)    ELSE BEGIN  (* b_i = 0 *)
(12)      IF DC ( k )  (* The TC_NORMAL(k) is represented from Targets[i+1] *)
(13)        THEN BEGIN
(14)               Child_Targets <-- Targets[ i + 1 ] TO Targets[ i + TLC( k ) ]
(15)               Child_MS <-- MS[ i + 1 ] TO MS[ i + TLC( k ) ]
(16)               IF NOT OutProcessor( k )
(17)                 THEN Targets_In ( k, Child_targets, Child_MS)
(18)                 ELSE BEGIN
(19)                        Targets[ k ] <-- Child_targets
(20)                        MS[ k ] <-- Child_MS
(21)                      END
(22)               i <-- i + TCL( k ) +1   (* Skip the total context *)
(23)             END
(24)        ELSE i <-- i + 1
(25)    END
(26)  END
```

Fig. 5.9 Searching targets for multisets $M_1(p)$ to $M_n(p)$

1. Self processing in membrane: Objects remaining inside membrane after evolution rules application will be sent to the father membrane together with $M_o(p)$

$$M_o(p) \leftarrow M_o(p) + w - M_a(p) + M_h(p)$$

   where w is the multiset of objects in the membrane at the beginning of the evolution step
2. Objects coming from other membranes in the current evolution step. In this case, it is needed to distinguish two possibilities depending on whether objects are processed by proxy: before or after proxy set the membrane as dissolved (field $D(j)$). .
   (a) $M(j)$ stores objects arriving the proxy before it has marked the membrane j as dissolved. At the moment the proxy marks membrane j as dissolved, it sends $M(j)$ to membrane j father using *Target_Out* algorithm. This behaviour is reached by adding lines 5 to 9 to the *ChangeUS* algorithm, as figure 5.10 shows.
   (b) In case of objects for membrane j arriving proxy after membrane j has been marked as dissolved and before the current evolution step has finished, the *Targets_In* algorithm will send them to membrane j father -lines 5 and 6 of figure 5.9-.

```
ChangeUS ( j , NewPerm, UState)
(1)   Mask <-- 1
(2)   IF NewPerm = Dissolved THEN BEGIN
(3)          Mask <-- Mask + UState
(4)          Mask <-- Mask  XOR (0 + USM ( j ))
               (* send up current USM of membrane j *)
(5)          D ( j ) <-- true
(6)          IF M( j ) <> null THEN BEGIN
(7)               Target_Out ( j, 1, M( j ))
(8)               M ( j ) <-- null
(9)        END
(10)  END
(11)  ……..
```

Fig. 5.10 In case of dissolution, $M(j)$ is sent to membrane j father

6 Results distribution

At this moment, when system proxies finish all the tasks explained above with algorithms *ChangeUS*, *Target_Out* and *Targets_In*, their results are stored in $USM(j)$, $M(j)$, $Targets(j)$ and $MS(j)$ fields, where membrane j may be allocated in other processor. In order to deliver these results, the distributed architecture in which the membrane system is implemented is very important. This is because of external communications are implemented by distributed architectures in different ways.

6.6 Architecture proposed by Tejedor et al. in [8] The external communications are established in depth in the processors tree. Therefore, after receiving information coming from the upper level, a processor P communicates with each descendant processor in both directions and from left to right; and finally, P sends data to its ascendant processor. Taking this order into account, the sequence of tasks to be carried out by processor P proxy is the following:

1. P proxy gets all data contained in $M(j)$, $Targets(j)$ and $MS(j)$ coming from ascendant processor proxy.
2. P proxy processes the arrived data using *Targets_In* algorithm for $Targets(j)$ and $MS(j)$ fields. Multisets received in $M(j)$ are placed in the corresponding $M(j)$ field, but if membrane j has been marked as dissolved in the current evolution step, then $M(j)$ has to be sent to membrane j father with *Target_Out* algorithm. In this case, $M(j)$ will come back to the ascendant processor in step 4.
3. for each descendant processor of P from left to right:
   (a) P proxy sends the corresponding information ($M(j)$, $Targets(j)$ and $MS(j)$) to the descendant processor.

   (b) P proxy waits until the descendant processor replies with data composed of fields $M(j)$ and $USM(j)$.
   (c) P proxy continues searching targets upwards from membrane j related to fields $M(j)$ and $USM(j)$ by using *Target_Out* and *ChangeUS* algorithms.
4. Once P proxy has processed all data from all its descendant processors, it sends to its ascendant processor the corresponding fields $M(j)$ and $USM(j)$.
5. Finally and through internal communications, P proxy delivers the definitive fields $M(j)$ and $USM(j)$ associated to inner membranes processor. The evolution step finishes when every membrane j updates its multiset and its usefulness state with this information, as follows:

$$w \leftarrow w + M(j)$$
$$Usefulness\,State \leftarrow Usefulness\,State\ XOR\ USM(j)$$

6.7 Architectures proposed by Bravo et al. in [2] and [3] These architectures make use of one [2] or several [3] master processors. Master processors are in charge of controlling communications among slaves processors, while membranes of the P system are placed on slaves processors. Hence, a master processor has to store and process $M(j)$, $USM(j)$, $TCL(j)$, $PFTC(j)$, $CD(j)$ and $D(j)$ fields, for all membranes belonging to slaves controlled by the master. As it was said above, $D(j)$ is a dynamic field and it is changed during execution by membranes. Therefore, a problem arises with the $D(j)$ field updating. Proxies associated to slave processors have to notify membranes dissolutions to master proxy.

The sequence of tasks in these architectures is the following:

1. Every slave processor proxy sends data to the suitable master in its corresponding turn. In particular, it sends fields $M(j)$, $USM(j)$, $Targets(j)$ and $MS(j)$ to its master, regardless of the target processor. Additionally, it has to send the list of dissolved membranes in the current evolution step.
2. Master proxy processes the incomming information as follows: $Target(j)$ and $MS(j)$ with *Targets_In* algorithm, $M(j)$ with *Target_Out* algorithm and $USM(j)$ with *ChangeUS* algorithm. Furthermore, master proxy updates $D(j)$ field for all dissolved membranes, taking into account the same questions as in section 5.3.
3. Master proxy sends the corresponding $M(j)$ and $USM(j)$ fields to each one of the slaves processors.
4. Finally, slave proxy sends to each one of its inner membranes their corresponding $M(j)$ and $USM(j)$ fields. Then, evolution step finishes.

7 Conclusion

Membranes make use of usefulness state to determine the set of membranes with which they can communicate. Moreover, when dissolutions or inhibitions are produced in the

system, it is only needed usefulness states changes in father membranes in order to reconfigure the membrane structure of P systems.

The work presented here shows that usefulness states can be implemented in several distributed architectures for membrane systems implementations [8], [2] and [3]. In addition, usefulness states solve permeability changes in membrane systems for the referred architectures.

In [5], membrane total context concept was defined. This paper shows how to use it in a very useful manner to encode targets in evolution rules, avoiding labels in membranes. This encoding method allows to find any target membrane in a precise way. Moreover, it can be used in several distributed architectures for membrane systems implementation [8], [2] and [3].

It is also presented here a semantic analysis of P systems for determining what kind of information is relevant in the phase of communications among membranes. In this sense, it was necessary to determine how to solve the targeting problem without producing an overload in the system communication; and how to update and communicate new membranes states all over the systems. The presented solution based on usefulness states has been proved to be useful at least in distributed architectures presented in [8], [2] and [3].

Bibliography

[1] F. Arroyo, C. Luengo, J. Castellanos, L.F. de Mingo, *A Binary Data Structure for Membrane Processors: Connectibity Arrays* (A. Alhazov, C. Martn-Vide and Gh. Păun Eds), 2003. Preproceedings of the Workshop on Membrane Computing, Tarragona, Spain, 41-52.

[2] G. Bravo, L. Fernández, F. Arroyo. J. Tejedor, *Master Slave Distributed Architecture for Membrane Systems Implementation*, 8th WSEAS Int. Conf. on Evolutionary Computing (EC'07), June 2007, Vancouver (Canada).

[3] G. Bravo, L. Fernández, F. Arroyo, M. A. Pea, *Hierarchical Master-Slave Architecture for Membrane Systems Implementation*, 13th Int. Symposium on Artificial Life and Robotics 2008 (AROB '08), Feb 2008, Beppu, Oitia (Japan).

[4] G.Ciobanu, W. Guo, *P Systems Running on a Cluster of Computers*. Workshop on Membrane Computing (Gh. Păun, G. Rozemberg, A. Salomaa Eds.),2004, LNCS 2933, Springer, 123-139

[5] J. A. Frutos, L. Fernández, F. Arroyo, G. Bravo, *Static Analysis of Usefulness States in Transition P Systems*, Proceedings of the Fifth International Conference, Information Research and Applications I.TECH 2007, June 2007, Varna, Bulgary. 174-182.

[6] Gh. Păun, *Computing with Membranes*, Journal of Computer and System Sciences, 61, 1, 108-143. 2000

[7] A.Syropoulos, E.G. Mamatas, P.C. Alliomes et al, *A Distributed Simulation of P Systems* (A. Alhazov, C. Martn-Vide and Gh. Păun Eds), 2003. Preproceedings of the Workshop on Membrane Computing, Tarragona, Spain, 455-460.

[8] J. Tejedor, L. Fernández, F. Arroyo, G.Bravo, *An Architecture for Attacking the Bottleneck Communication in P Systems*, In: M. Sugisaka, H. Tanaka (eds.), Proceedings of the 12th Int. Symposium on Artificial Life and Robotics, Jan 25-27, 2007, Beppu, Oita, Japan, 500-505.

A P-Lingua Programming Environment for Membrane Computing

Daniel Díaz–Pernil, Ignacio Pérez–Hurtado, Mario J. Pérez–Jiménez, Agustín Riscos–Núñez.

University of Sevilla, Dpt. Computer Science and Artificial Intelligence,
Avda. Reina Mercedes s/n. 41012 Sevilla, Spain
{sbdani,perezh,marper,ariscosn}@us.es

A new programming language for membrane computing, P-Lingua, is developed in this paper. This language is not designed for a specific simulator software. On the contrary, its purpose is to offer a general syntactic framework that could define a unified standard for membrane computing, covering a broad variety of models. At the present stage, P-Lingua can only handle P systems with active membranes, although the authors intend to extend it to other models in the near future.
P-Lingua allows to write programs in a friendly way, as its syntax is very close to standard scientific notation, and parameterized expressions can be used as shorthand for sets of rules. There is a built-in compiler that parses these human-style programs and generates XML documents that can be given as input to simulation tools, different plugins can be designed to produce specific adequate outputs for existing simulators.
Furthermore, we present in this paper an integrated development environment that plays the role of interface where P-lingua programs can be written and compiled. We also present a simulator for the class of recognizer P systems with active membranes, and we illustrate it by following the writing, compiling and simulating processes with a family of P systems solving the SAT problem.

1 Introduction

Membrane computing (or cellular computing) is an emerging branch of Natural Computing that was introduced by Gh. Păun [5]. The main idea is to consider biochemical processes taking place inside living cells from a computational point of view, in a way that provides a new nondeterministic model of computation.

The initial definition of this computing paradigm is very flexible, and many different models have been defined and investigated in the area: P systems with symport/antiport rules, with active membranes, with catalysts, with promoters/inhibitors, etc. There were some attempts to establish a common formalization covering most of the existing models (see e.g. [2]), but the membrane computing community is still using specific syntax and semantics depending on the model they work with.

This diversification also exists in what concerns the development of software applications for the simulation of P systems (see [3], [11]), as such applications are usually focused on, and adapted for, particular cases, making it difficult to work on generalizations.

It is convenient to unify standards (specifications that regulate the performance of specific processes in order to guarantee their interoperability) and to implement the necessary tools and libraries in order to give the first steps towards a next generation of applications.

When designing software for membrane computing, one has to precisely describe the variant specification that is to be simulated. This task is hard if we need to handle families of P systems where the set of rules, the alphabet, the initial contents and even the membrane structure depend on the value assigned to some initial parameters. In existing software, several options have been implemented: plain text files with a determined format, XML documents, graphical user interfaces, etc. As mentioned above, most of these solutions are adapted to specific models or to the specific purpose of the software.

In this paper we propose a programming language, called P-Lingua, whose programs define P systems in a parametric and modular way. After assigning values to the initial parameters, the compilation tool generates an XML document associated with the corresponding P system from the family, and furthermore it checks possible programming errors (both lexical/syntactical and semantical). Such documents can be integrated into other applications, thus guaranteeing interoperability. More precisely, in the simulators framework, the XML specification of a P system can be translated into an executable representation.

We present a practical application of P-Lingua giving a simulator for recognizer P systems with active membranes that receives as input an XML document generated by the compiler and that allows us to simulate a computation, obtaining the correct answer of the system (due to the confluence of it), and a text file with a detailed step-by-step report of the computation. We also show an integrated development environment that plays the role of interface where P-Lingua programs can be written and compiled.

The paper is structured as follows. In Section 2 several definitions and concepts are given for the sake of selfcontainment of the paper. The next section introduces the P-Lingua programming language, and the syntax for P systems with active membranes is specified. A solution to the SAT problem using P-Lingua is implemented in Section 4. The compilation tool for the language is presented in the next section. In Section 6 we present an integrated development environment for P-Lingua. Section 7 presents a simulator for recognizer P systems with active membranes. Finally, some conclusions and ideas for future work are presented.

2 Preliminaries

Polynomial time solutions to computationally hard problems in membrane computing are achieved by trading time for space. This is inspired by the capability of cells to produce an exponential number of new membranes in linear time. There are many ways a living cell can produce new membranes: *mitosis* (cell division), *autopoiesis* (membrane creation), *gemmation*, etc. Following these inspirations a number of different variants of P systems has arisen, and many of them proved to be computationally universal.

For the sake of simplicity, we shall focus in this paper on a model, *P systems with active membranes*. It is a construct of the form $\Pi = (O, H, \mu, \omega_1, \ldots, \omega_m, R)$, where $m \geq 1$ is the initial degree of the system; O is the alphabet of *objects*, H is a finite set of *labels* for membranes; μ is a membrane structure, consisting of m membranes injectively labelled with elements of H, $\omega_1, \ldots, \omega_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ; and R is a finite set of *rules*, where each rule is of one of the following forms:

(a) $[a \to v]_h^\alpha$ where $h \in H$, $\alpha \in \{+, -, 0\}$ (electrical charges), $a \in O$ and v is a string over O describing a multiset of objects associated with membranes and depending on the label and the charge of the membranes (*object evolution rules*);

(b) $a\,[\ ]_h^\alpha \to [b]_h^\beta$ where $h \in H$, $\alpha, \beta \in \{+, -, 0\}$, $a, b \in O$ (*send-in communication rules*). An object is introduced in the membrane, possibly modified, and the initial charge α is changed to β;

(c) $[a]_h^\alpha \to [\ ]_h^\beta b$ where $h \in H$, $\alpha, \beta \in \{+, -, 0\}$, $a, b \in O$ (*send-out communication rules*). An object is sent out of the membrane, possibly modified, and the initial charge α is changed to β;

(d) $[a]_h^\alpha \to b$ where $h \in H$, $\alpha \in \{+, -, 0\}$, $a, b \in O$ (*dissolution rules*). A membrane with a specific charge is dissolved in reaction with a (possibly modified) object;

(e) $[a]_h^\alpha \to [b]_h^\beta\,[c]_h^\gamma$ where $h \in H$, $\alpha, \beta, \gamma \in \{+, -, 0\}$, $a, b, c \in O$ (*division rules*). A membrane is divided into two membranes. The objects inside the membrane are replicated, except for a, that may be modified in each membrane.

Rules are applied according to the following principles:

- All the elements which are not involved in any of the operations to be applied remain unchanged.
- Rules associated with label h are used for all membranes with this label, no matter whether the membrane is an initial one or whether it was generated by division during the computation.
- Rules from (a) to (e) are used as usual in the framework of membrane computing, i.e. in a maximal parallel way. In one step, each object in a membrane can only be used by at most one rule (non-deterministically chosen), but any object which can evolve by a rule must do it (with the restrictions indicated below).

- Rules (b) to (e) cannot be applied simultaneously in a membrane in one computation step.
- An object a in a membrane labelled with h and with charge α can trigger a division, yielding two membranes with label h, one of them having charge β and the other one having charge γ. Note that all the contents present before the division, except for object a, can be the subject of rules in parallel with the division. In this case we consider that in a single step two processes take place: "first" the contents are affected by the rules applied to them, and "after that" the results are replicated into the two new membranes.
- If a membrane is dissolved, its content (multiset and interior membranes) becomes part of the immediately external one. The skin is never dissolved.

The so-called recognizer P systems were introduced in [6], and constitute the natural framework to study the solvability of decision problems, since deciding whether an instance has an affirmative or negative answer is equivalent to deciding if a string belongs or not to the language associated with the problem.

In the literature, recognizer P systems are associated in a natural way with P systems with *input*. The data related to an instance of the decision problem has to be provided to the P system in order for it to compute the appropriate answer. This is done by codifying each instance as a multiset placed in an *input membrane*. The output of the computation, yes or no, is sent to the environment.

A *P system with input* is a tuple (Π, Σ, i_{Π}), where: (a) Π is a P system, with working alphabet Γ, with p membranes labelled by $1, \ldots, p$, and initial multisets $\omega_1, \ldots, \omega_p$ associated with them; (b) Σ is an (input) alphabet strictly contained in Γ; the initial multisets are over $\Gamma \setminus \Sigma$; and (c) i_{Π} is the label of a distinguished (input) membrane.

For each multiset, m, over Σ, the *initial configuration* of (Π, Σ, i_{Π}) with input m is $(\mu, \omega_1, \ldots, \omega_{i_{\Pi}} + m, \ldots, \omega_p)$.

A *recognizer P system* is a P system with input, (Π, Σ, i_{Π}), and with external output such that:

(a) The working alphabet contains two distinguished elements, yes and no.

(b) All computations halts.

(c) If $\mathcal{C}$ is a computation of Π, then either the object yes or the object no (but no both) must have been released into the environment, and only in the last step of the computation.

We say that $\mathcal{C}$ is an accepting computation (respectively, rejecting computation) if the object yes (respectively, no) appears in the external environment associated with the corresponding halting configuration of C.

3 The P-Lingua programming language

A programming language is an artificial language that can be used to control the behavior of a machine, particularly a computer, but it can be used also to define a model of a machine that can be translated into an executable representation by a simulation tool.

Programming languages are defined by syntactic and semantic rules which describe their structure and their meaning, respectively.

The P-Lingua programming language intends to define a broad variety of P system models. At the present stage, P-Lingua can only define P systems with active membranes, but other models will be added to the language specification in future works.

What follows is the syntax of the language for P systems with active membranes (originally presented at [1]).

3.1 Valid identifiers We say that a sequence of characters forms a `valid identifier` if it does not begin with a numeric character and it is composed by characters from the following:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
  0 1 2 3 4 5 6 7 8 9 _
```

Valid identifiers are widely used in the language: to define module names, parameters, indexes, membrane labels and alphabet objects.

The following text strings are reserved words in the language: `def, call, @mu, @ms, main, -->, #` and they cannot be used as valid identifiers.

3.2 Identifiers for electrical charges In P-Lingua, we can consider electrical charges by using the + and - symbols for positive and negative charges, respectively, and no one for neutral charge. It is worth mentioning that polarizationless P systems are included.

3.3 Variables Two kind of variables are permitted in P-Lingua:

- `indexes`
- `Parameters`

Variables are used to store numeric values and their names are valid identifiers. We use 32 bits (signed), this allows a range from -2^{31} to $2^{31} - 1$.

3.4 Numeric expressions Numeric expressions can be written by using the `*` (multiplication), `/` (division), `%` (modulo), `+` (addition), `-` (subtraction) operators with integer numbers or variables, along with the use of parentheses.

3.5 Objects The objects of the alphabet of a P system are written using valid identifiers, and the inclusion of sub-indexes is permitted. For example, $x_{i,2n+1}$ and Yes are written as `x{i,2*n+1}` and `Yes`, respectively.

The multiplicity of an object is represented by using the `*` operator. For example, x_i^{2n+1} is written as `x{i}*(2*n+1)`.

3.6 Modules definition Similarities between various solutions to **NP**-complete numerical problems by using families of recognizer P systems are discussed in [4]. Also, a cellular programming language is proposed based on libraries of subroutines. Using these ideas, a P-Lingua program consists of a set of programming modules that can be used more times by the same, or other, programs.

The syntax to define a module is the following.

```
def module_name(param1,..., paramN)
{
  sentence0;
  sentence1;
  ...
  sentenceM;
}
```

The name of a module, `module_name`, must be a valid and unique identifier. The parameters must be valid identifiers and cannot appear repeated. It is possible to define a module without parameters. Parameters have a numerical value that is assigned at the module call (see below).

All programs written in P-Lingua must contain a `main` module without parameters. The compiler will look for it when generating the XML file.

In P-Lingua there are sentences to define the membranes configuration of a P system, to specify multisets, to define rules and to make calls to other modules. Next, let us see how such sentences are written.

3.7 Module calls In P-Lingua, modules are executed by using calls. The format of a sentence that calls a module for some specific values of its parameters is given next:

```
call module_name(value1, ..., valueN);
```

where `value`i is an integer number or a variable.

3.8 Definition of the initial membrane structure of a P system In order to define the initial membrane structure of a P system, the following sentence must be written:

```
@mu = expr;
```

where `expr` is a sequence of matching square brackets representing the membrane structure, including some identifiers that specify the label and the electrical charge of each membrane.

Examples:

1. $[[\,]^0_2]^0_1 \equiv$ `@mu` $=$ `[[]'2]'1`
2. $[[\,]^0_b[\,]^-_c]^+_a \equiv$ `@mu` $=$ `+[[]'b, -[]'c]'a`

3.9 Definition of multisets The next sentence defines the initial multiset associated to the membrane labelled by `label`.

```
@ms(label) = list_of_objects;
```

where `label` is a valid identifier or a natural number that represents a label of the structure of membranes and `list_of_objects` is a comma-separated list of objects. The character # is used to represent the empty multiset.

3.10 Union of multisets P-Lingua allows to define the union of two multisets (recall that the input multiset is *added* to the initial multiset of the input membrane) by using a sentence with the following format.

```
@ms(label) += list_of_objects;
```

3.11 Definition of rules

1. The format to define *evolution rules* of type $[a \rightarrow v]^\alpha_h$ is given next:
   α`[a --> v]'h`
2. The format to define *send-in communication rules* of type $a\,[\,]^\alpha_h \rightarrow [b]^\beta_h$ is given next:
   aα`[]'h -->`β`[b]`
3. The format to define *send-out communication rules* of type $[a]^\alpha_h \rightarrow b[\,]^\beta_h$ is given next:

α`[a]'h -->` β`[]b`

4. The format to define *division rules* of type $[a\,]_h^\alpha \rightarrow [b]_h^\beta[c]_h^\gamma$ is given next:
   α`[a]'h -->`β`[b]`γ`[c]`
5. The format to define *dissolution rules* of type $[a]_h^\alpha \rightarrow b$ is given next:
   α`[a]'h --> b`

where:

- α, β and γ are identifiers for electrical charges;
- `a`, `b` and `c` are objects of the alphabet;
- `v` is a comma-separated list of objects that represents a multiset;
- `h` is a label (the symbol `'` always precedes a label name).

Some examples:

- $[x_{i,1} \rightarrow r_{i,1}^4]_2^+ \equiv$ `+[x{i,1} --> r{i,1}*4]'2`
- $d_k[\,]_2^0 \rightarrow [d_{k+1}]_2^0 \equiv$ `d{k}[]'2 --> [d{k+1}]`
- $[d_k]_2^+ \rightarrow []_2^0 d_k \equiv$ `+[d{k}]'2 --> []d{k}`
- $[d_k]_2^0 \rightarrow [d_k]_2^+[d_k]_2^- \equiv$ `[d{k}]'2 --> +[d{k}]-[d{k}]`
- $[a]_2^- \rightarrow b \equiv$ `-[a]'2 --> b`

3.12 Parametric sentences In P-Lingua, it is possible to define parametric sentences by using the next format:

```
sentence : range1, ..., rangeN;
```

where `sentence` is a sentence of the language, or a sequence of sentences in brackets, and `range1, ..., rangeN` is a comma-separated list of ranges with the format:

```
min_value <= index <= max_value
```

where `min_value` and `max_value` are numeric expressions, integer numbers or variables, and `index` is a variable that can be used in the context of the sentence. It is possible to use the operator $<$ instead of $<=$.

The sentence will be repeated for each possible values of each `index`.

Some examples of parametric sentences:

1. $[d_k]_2^0 \rightarrow [d_k]_2^+[d_k]_2^- : 1 \leq k \leq n \equiv$
   `[d{k}]'2 --> +[d{k}]-[d{k}] : 1<= k <= n;`
2. $[x_{i,j} \rightarrow x_{i,j-1}]_2^+ : 1 \leq i \leq m, 2 \leq j \leq n \equiv$
   `+[x{i,j} --> x{i,j-1}]'2 : 1<=i<=m,2<=j<=n;`

3.13 Inclusion of comments The programs in P-Lingua can be commented by writing phrases into the text strings `/*` and `*/`.

4 Implementation of a solution to SAT

In this section, we present a solution to the **SAT** problem using recognizer P systems with active membranes, given by M.J. Pérez–Jiménez et al. [7].

For each $(m, n) \in \mathbb{N}^2$, we consider the P system $(\Pi(\langle m, n\rangle), \Sigma(m, n), i(m, n))$, where

- $\Sigma(m,n) = \{x_{i,j}, \bar{x}_{i,j} : 1 \leq i \leq m,\ 1 \leq j \leq n\}$
- $i(m,n) = 2$
- $\Pi(\langle m, n\rangle) = (\Gamma(m,n), \{1,2\}, [\ ]_2\ ]_1, w_1, w_2, R)$, is defined as follows:
  - $\Gamma(m,n) = \Sigma(m,n) \cup \{c_k : 1 \leq k \leq m+2\} \cup$
    $\{d_k : 1 \leq k \leq 3n+2m+3\} \cup$
    $\{r_{i,k} : 0 \leq i \leq m,\ 1 \leq k \leq m+2\} \cup \{e, t\} \cup \{Yes, No\}$
  - $w_1 = \emptyset$
  - $w_2 = \{d_1\}$
  - The set of rules, R, is given by:

$\{[d_k]_2^0 \rightarrow [d_k]_2^+[d_k]_2^- : 1 \leq k \leq n\}$

$\{[x_{i,1} \rightarrow r_{i,1}]_2^+, [\bar{x}_{i,1} \rightarrow r_{i,1}]_2^- : 1 \leq i \leq m\}$

$\{[x_{i,1} \rightarrow \lambda]_2^-, [\bar{x}_{i,1} \rightarrow \lambda]_2^+ : 1 \leq i \leq m\}$

$\{[x_{i,j} \rightarrow x_{i,j-1}]_2^+, [x_{i,j} \rightarrow x_{i,j-1}]_2^- : 1 \leq i \leq m, 2 \leq j \leq n\}$

$\{[\bar{x}_{i,j} \rightarrow \bar{x}_{i,j-1}]_2^+, [\bar{x}_{i,j} \rightarrow \bar{x}_{i,j-1}]_2^- : 1 \leq i \leq m, 2 \leq j \leq n\}$

$\{[d_k]_2^+ \rightarrow [\,]_2^0 d_k, [d_k]_2^- \rightarrow [\,]_2^0 d_k : 1 \leq k \leq n\}$

$\{d_k[\,]_2^0 \rightarrow [d_{k+1}]_2^0 : 1 \leq k \leq n-1\}$

$\{[r_{i,k} \rightarrow r_{i,k+1}]_2^0 : 1 \leq i \leq m, 1 \leq k \leq 2n-1\}$

$\{[d_k \rightarrow d_{k+1}]_1^0 : n \leq k \leq 3n-3\};\ [d_{3n-2} \rightarrow d_{3n-1}e]_1^0$

$e[\,]_2^0 \rightarrow [c_1]_2^+; [d_{3n-1} \rightarrow d_{3n}]_1^0$

$\{[d_k \rightarrow d_{k+1}]_1^0 : 3n \leq k \leq 3n + 2m + 2\}$

$[r_{1,2n}]_2^+ \rightarrow []_2^- r_{1,2n}\ ;\ \{[r_{i,2n} \rightarrow r_{i-1,2n}]_2^- : 1 \leq i \leq m\}$

$r_{1,2n}[\,]_2^- \rightarrow [r_{0,2n}]_2^+$

$\{[c_k \rightarrow c_{k+1}]_2^- : 1 \leq k \leq m\}$

$[c_{m+1}]_2^+ \rightarrow []_2^+ c_{m+1}\ ;\ [c_{m+1} \rightarrow c_{m+2}t]_1^0$

$[t]_1^0 \rightarrow []_1^+ t\ ;\ [c_{m+2}]_1^+ \rightarrow []_1^- Yes\ ;\ [d_{3n+2m+3}]_1^0 \rightarrow []_1^+ No$

4.14 Implementation The following is the code of the program written in P-Lingua that specifies a family of P systems solving the **SAT** problem.

Objects of the form $\bar{x}_{i,j}$ are written as `nx{i,j}`.

```
/* Module that defines a family of recognizer P systems
   to solve the SAT problem */
def Sat(m,n)
{
 /* Initial configuration */
 @mu = [[]'2]'1;

 /* Initial multisets */
 @ms(2) = d{1};

 /* Set of rules */
 [d{k}]'2 --> +[d{k}]-[d{k}] : 1 <= k <= n;

 {
  +[x{i,1} --> r{i,1}]'2;
  -[nx{i,1} --> r{i,1}]'2;
  -[x{i,1} --> #]'2;
  +[nx{i,1} --> #]'2;
 } : 1 <= i <= m;

 {
  +[x{i,j} --> x{i,j-1}]'2;
  -[x{i,j} --> x{i,j-1}]'2;
  +[nx{i,j} --> nx{i,j-1}]'2;
  -[nx{i,j} --> nx{i,j-1}]'2;
 } : 1<=i<=m, 2<=j<=n;

 {
  +[d{k}]'2 --> []d{k};
  -[d{k}]'2 --> []d{k};
 } : 1<=k<=n;

 d{k}[]'2 --> [d{k+1}] : 1<=k<=n-1;
```

```
 [r{i,k} --> r{i,k+1}]'2 : 1<=i<=m, 1<=k<=2*n-1;
 [d{k} --> d{k+1}]'1 : n <= k<= 3*n-3;
 [d{3*n-2} --> d{3*n-1},e]'1;
 e[]'2 --> +[c{1}];
 [d{3*n-1} --> d{3*n}]'1;
 [d{k} --> d{k+1}]'1 : 3*n <= k <= 3*n+2*m+2;
 +[r{1,2*n}]'2 --> -[]r{1,2*n};
 -[r{i,2*n} --> r{i-1,2*n}]'2 : 1<= i <= m;
 r{1,2*n}-[]'2 --> +[r{0,2*n}];
 -[c{k} --> c{k+1}]'2 : 1<=k<=m;
 +[c{m+1}]'2 --> +[]c{m+1};
 [c{m+1} --> c{m+2},t]'1;
 [t]'1 --> +[]t;
 +[c{m+2}]'1 --> -[]Yes;
 [d{3*n+2*m+3}]'1 --> +[]No;

} /* End of Sat module */

/* Main module */
def main()
{
 /* Call to Sat module for m=4 and n=6 */
 call Sat(4,6);
 /* Expansion of the input multiset */
 @ms(2) += x{1,1}, nx{1,2}, nx{2,2}, x{2,3},
           nx{2,4}, x{3,5}, nx{4,6};
} /* End of main module */
```

The module `main` is instantiated with the formula

$$\varphi \equiv (x_1 + \overline{x}_2)(\overline{x}_2 + x_3 + \overline{x}_4)\, x_5\, \overline{x}_6$$

where $n = 6$, $m = 4$ and the input multiset: $x_{1,1}, \overline{x}_{1,2}, \overline{x}_{2,2}, x_{2,3}, \overline{x}_{2,4}, x_{3,5}, \overline{x}_{4,6}$.

5 The compilation tool

Programming languages are associated with compilation tools, which are computer programs that translate text written in a programming language into another language. The original text is usually called the *source code* whereas the output is called the *object code*. Commonly the output has a form suitable for being processed by other programs or for being executed by the computer, but it may as well be a human-readable text file.

We have developed a compilation tool that is able to translate programs written in P-lingua into XML documents, after having assigned values to some initial parameters. Moreover, plugins can be designed and added to produce object code with different formats.

Recall that a P-lingua program can encode a family of P systems (with the help of some parameters) in a flexible manner, whereas the object code generated by the compilation tool specifies only a single P system of the family. In this way, the applications that accept that object code do not need to process parametric systems, and hence their implementation is much easier.

The e**X**tensible **M**arkup **L**anguage (XML) is a general-purpose specification for creating custom markup languages. It is classified as an extensible metalanguage because it allows the users to define their own elements. Its primary purpose is to facilitate the sharing of structured data across different information systems. It is worth mentioning that the SBML (Systems Biology Markup Language) is a XML language encoding the main components of biochemical networks. It is used by several existing simulators for P systems (see the software link at [11]).

The complete syntax of the XML language generated by the compilation tool for P systems with active membranes can be found at [1].

The tool may be executed from the command line as follows:

```
plingua input_file -xml output_file [-v verbosity_level] [-h]
```

The text file `input_file` contains the program (written in P-lingua) that we want to be compiled, and `output_file` is the name of the XML file that is generated. Optional arguments are in brackets: the option `-v verbosity_level` is a number between 0 and 5 indicating the level of detail of the messages shown during the compilation process, and the option `-h` displays some help information.

6 An integrated development environment

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. Usually, an IDE consists of a source code editor, a compiler and/or interpreter, a debugger, and other useful tools.

Typically an IDE is devoted to a specific programming language, so as to provide a feature set which most closely matches the programming paradigms of the language. In this sense, we have developed an IDE for P-Lingua by using the Java language. This application provides an environment to write and debug programs in P-Lingua for P systems with active membranes, and it can be updated by adding plugins to accept future versions of the language. The IDE can also be used as a simulation tool for P-Lingua programs.

This application includes a source code editor with syntax highlighting which is a feature that displays text source code in different colors and fonts, as both structures and

syntax errors are visually distinct. With this editor, it is also possible to generate P-Lingua programs composed of several single files.

A compilation tool is included to check possible programming errors and to generate XML files that can be used in third-part applications.

A simulation tool for debugging is included in order to aid the researcher in the task of designing new P systems. This tool provides simulations by using an interactive step-by-step mode. The user can choose between simulation of one or several steps, or let the simulation run until a halting state. A lot of information is given in each step of the simulation: a tree-view of the membranes structure, complete information of the multisets and the set of rules selected to be executed. The user can also choose between different non-deterministic ways of computation, or let the software select one.

7 A simulator for recognizer P systems with active membranes

The act of simulating generally entails representing certain key characteristics or behaviors of some physical, or abstract, system. We must distinguish a simulation tool from an emulation tool: this duplicates the functions of one system by using a different system, so that the second system behaves like (and appears to be) the emulated system. With the current technology, we cannot emulate the functionality of a cellular machine (a membrane system) by using a conventional computer to solve instances of **NP**-hard problems in a polynomial time, but we can simulate these cellular machines for research purposes, even if the simulation is not done in a polynomial time.

The P system computations are massively parallel. One of the most common programming methods to simulate real parallelism in a conventional computer with a single processor is to use multithreading. A thread in this sense is a thread of execution. Threads are a way for a program to fork (or split) itself into two or more simultaneously (or pseudo-simultaneously) running tasks. Multiple threads can be executed in parallel on a single computer. This multithreading generally occurs by time-division multiplexing where the processor switches between different threads. This context switching can happen so fast as to give the illusion of parallelism to an end-user. On a multiprocessor or multi-core system, threading can be achieved via multiprocessing, wherein different threads can literally run simultaneously on different processors or cores.

As a first practical application of the P-lingua programming language, we have implemented a simulator for recognizer P systems with active membranes that takes as input an XML document generated by the P-lingua compiler and runs one of the possible computations that the P system may follow, obtaining the answer that the system outputs to its environment, and a text file with a detailed step-by-step report of the computation.

The system requirements are the same as in the case of the P-lingua compiler.

The simulator is launched from the command line as follows:

```
plingua_sim input_xml [-o output_file]
```

where `input_xml` is an XML document formatted as discussed in this paper, and `output_file` is the name of the file where the report about the simulated computation will be saved.

7.15 Simulation of a solution to the SAT problem We now show an execution of the simulator running on the XML document obtained after compiling the P-lingua program described in Section 4.14. The results have been obtained on an AMD Sempron machine, at 2.8 Ghz and with 512Mb of RAM memory.

The command used to execute the simulation is:

```
plingua_sim sat.xml -o info.txt
```

The simulation ends when no more rules can be applied, and then the following information is displayed:

```
Environment multiset: t, Yes
   Steps: 41
   Time: 1.971 s.
   Halting configuration (No rule can be selected to be executed
in the next step)
```

Thus, the computation of the P system spend 41 transition steps, and it took 1.971 seconds to simulate it until reaching a halting configuration (recall that we are simulating a parallel device on a sequential computer).

The file `info.txt` keeps detailed information about each configuration of the simulated computation. More precisely, the multisets and polarizations of all the membranes are listed, as well as the rules selected for execution at each transition step. The configurations are numbered (starting at 0), to keep track of the step of the computation that is being simulated. Some information about the CPU time is shown for each step, and the number of rules of each type that is executed. As an example, we give the information generated for the first two configurations.

```
### MEMBRANE ID: 1, Label: 2, Charge: 0
    Multiset: nx{1, 2}, d{1}, x{3, 5}, nx{2, 4}, nx{2, 2},
              nx{4, 6}, x{2, 3}, x{1, 1}
    Parent Membrane ID: 0
    Rules Selected:
    1*DIVISION RULE: [d{1}]'2 --> +[d{1}] -[d{1}]
```

```
@@@ SKIN MEMBRANE ID: 0, Label: 1, Charge: 0
    Multiset: #
    Internal membranes count: 1

Configuration: 0
Time: 0.0 s.
1 division rule(s) selected to be executed in the step 1
**********************************************
### MEMBRANE ID: 1, Label: 2, Charge: +
    Multiset: nx{1, 2}, d{1}, x{3, 5}, nx{2, 4}, nx{2, 2},
              nx{4, 6}, x{2, 3}, x{1, 1}
    Parent Membrane ID: 0
    Rules Selected:
    1*EVOLUTION RULE: +[nx{2, 2} --> nx{2, 1}]'2
    1*EVOLUTION RULE: +[nx{1, 2} --> nx{1, 1}]'2
    1*EVOLUTION RULE: +[x{3, 5} --> x{3, 4}]'2
    1*EVOLUTION RULE: +[x{1, 1} --> r{1, 1}]'2
    1*EVOLUTION RULE: +[nx{2, 4} --> nx{2, 3}]'2
    1*EVOLUTION RULE: +[nx{4, 6} --> nx{4, 5}]'2
    1*EVOLUTION RULE: +[x{2, 3} --> x{2, 2}]'2
    1*SEND-OUT RULE: +[d{1}]'2 --> []d{1}

### MEMBRANE ID: 2, Label: 2, Charge: -
    Multiset: nx{1, 2}, d{1}, nx{2, 4}, x{3, 5}, nx{2, 2},
              x{2, 3}, nx{4, 6}, x{1, 1}
    Parent Membrane ID: 0
    Rules Selected:
    1*EVOLUTION RULE: -[nx{2, 4} --> nx{2, 3}]'2
    1*EVOLUTION RULE: -[nx{2, 2} --> nx{2, 1}]'2
    1*EVOLUTION RULE: -[nx{4, 6} --> nx{4, 5}]'2
    1*EVOLUTION RULE: -[x{1, 1} --> #]'2
    1*EVOLUTION RULE: -[x{2, 3} --> x{2, 2}]'2
    1*EVOLUTION RULE: -[nx{1, 2} --> nx{1, 1}]'2
    1*EVOLUTION RULE: -[x{3, 5} --> x{3, 4}]'2
    1*SEND-OUT RULE: -[d{1}]'2 --> []d{1}

@@@ SKIN MEMBRANE ID: 0, Label: 1, Charge: 0
    Multiset: #
    Internal membranes count: 2

Configuration: 1
Time: 0.025 s.
14 evolution rule(s) selected to be executed in the step 2
2 send-out rule(s) selected to be executed in the step 2
**********************************************
```

After simulating 41 transition steps, the halting configuration is described as follows:

```
### MEMBRANE ID: 1, Label: 2, Charge: +
    Multiset: r{0, 12}*3, c{4}
    Parent Membrane ID: 0
```

```
### MEMBRANE ID: 2, Label: 2, Charge: +
    Multiset: c{1}, r{2, 12}, r{3, 12}
    Parent Membrane ID: 0

### MEMBRANE ID: 3, Label: 2, Charge: +
    Multiset: r{0, 12}*5, c{4}
    Parent Membrane ID: 0

### MEMBRANE ID: 4, Label: 2, Charge: +
    Multiset: r{0, 12}*4, c{4}
    Parent Membrane ID: 0

### MEMBRANE ID: 5, Label: 2, Charge: +
    Multiset: r{0, 12}, r{2, 12}, c{2}
    Parent Membrane ID: 0

### MEMBRANE ID: 6, Label: 2, Charge: +
    Multiset: c{1}, r{3, 12}
    Parent Membrane ID: 0

### MEMBRANE ID: 7, Label: 2, Charge: +
    Multiset: 4*r{0, 12}, c{4}
    Parent Membrane ID: 0
```

⋮

```
@@@ SKIN MEMBRANE ID: 0, Label: 1, Charge: -
    Multiset: t*10, d{29}*64, c{6}*10
    Internal membranes count: 64

~~~ENVIRONMENT: t, Yes

Configuration 41
Time: 1.971 s.
Halt configuration (No rule can be selected to be
executed in the next step)

**********************************************
```

Note that there are 64 different membranes labelled by 2 in this configuration, although for the sake of simplicity we show only seven of them.

8 Conclusions and future work

In this paper we have presented a programming language for membrane computing, *P-Lingua*, together with a compiler that generates XML documents, an integrated development environment and a simulator for a class of P systems, namely recognizer P systems with active membranes.

Using a programming language to define cellular machines is a concept in the development of applications for membrane computing that leads to a standardization with the following advantages:

- Users can define cellular machines in a modular and parametric way by using an easy-to-learn programming language.
- It is possible to define libraries of modules that can be shared among users to facilitate the design of new programs.
- This method to define P systems is decoupled from its applications and the same P-Lingua programs can be used in different software environments.
- By using compiling tools, the P-Lingua programs are translated to other file formats that can be interpreted by a large number of different applications.

The first version of P-Lingua is presented for P systems with active membranes. In forthcoming versions, we will try to generalize the language so that other types of cellular devices can be also defined, for instance transition P systems and tissue P systems. In this sense, necessary plugins (software modules) for the IDE and the compilation tool must be developed also.

We have chosen an XML language as the output format because of the reasons exposed above. However, we are aware that for some applications it is not the most suitable format, due to the fact that XML does not include any method for compressing data, and therefore the text files can eventually become too large, which is a clear disadvantage for applications running on networks of processors. It would be convenient to improve the compiler (by adding plugins) so that it generates a larger variety of output formats, of special interest are compressed binary files or executable code (either in C or Java).

It is important to recall that the simulator presented here is designed to run in a conventional computer, having limited resources (RAM, CPU), and this leads to a bound on the size of the instances of computationally hard problems whose solutions can be successfully simulated. Moreover, conventional computers are not massively parallel devices, and therefore it seems that the inherent parallelism of P systems must be simulated by means of multithreading techniques. It would be interesting to design heuristics which help us to find good (short) computations.

These shortcomings lead us to the possibility of implementing a distributed simulator running on a network or cluster of processors, where the need of resources arising during the computation could be solved by adding further nodes to the network, thus moving towards massive parallelism.

The sofware presented in this paper and its source code can be freely downloaded from the *software* section on the website [12]. This software is under the GNU General Public License (GNU GPL) [8] and it is written in Java [9] using the lexical and syntactical

analyzers provided by JavaCC [10]. The minimum system requirements are having a Java virtual machine (JVM) version 1.6.0 running in a Pentium III computer.

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

Bibliography

[1] D. Díaz–Pernil, I. Pérez–Hurtado, M.J. Pérez–Jiménez, A. Riscos–Núñez. P-Lingua: A programming Language for Membrane Computing. In Daniel Díaz–Pernil, M.A. Gutiérrez–Naranjo, C. Graciani–Díaz, Gh. Păun, I. Pérez–Hurtado, A. Riscos–Núñez (eds) *Proceedings of the 6th Brainstorming Week on Membrane Computing, Sevilla*, Fénix Editora, (2008), 135–155.

[2] R. Freund, S. Verlan. A Formal Framework for Static (Tissue) P Systems. *Lecture Notes in Computer Science*, **4860** (2007), 271-284.

[3] M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez. Available Membrane Computing Software. In G. Ciobau, Gh. Păun, M.J. Péréz–Jiménez (eds.) *Applications of Membrane Computing, Natural Computing Series*, Springer–Verlag, 2006. Chapter **15** (2006), pp. 411–436.

[4] M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez. Towards a programming language in cellular computing. *Electronic Notes in Theoretical Computer Science*, Elsevier B.V., **123** (2005), 93–110.

[5] 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.

[6] M.J. Pérez-Jiménez, A. Romero–Jiménez, F. Sancho–Caparrini. Complexity classes in cellular computing with membranes. *Natural Computing*, **2**, 3 (2003), 265-285.

[7] M.J. Pérez–Jiménez, A. Romero–Jiménez, F. Sancho–Caparrini. A polynomial complexity class in P systems using membrane division. *Journal of Automata, Languages and Combinatorics*, 11, 4 (2006), 423-434.

[8] The GNU General Public License: `http://www.gnu.org/copyleft/gpl.html`

[9] Java web page: `http://www.java.com/`

[10] JavaCC web page: `https://javacc.dev.java.net/`

[11] P systems web page: `http://ppage.psystems.eu/`

[12] Research Group on Natural Computing web page: `http://www.gcn.us.es/`

On Testing P Systems

Marian Gheorghe[1], Florentin Ipate[2]

[1]The University of Sheffield, Department of Computer Science,
Regent Court, Portobello Street, Sheffield S1 4DP, UK
M.Gheorghe@dcs.shef.ac.uk

[2]The University of Pitesti, Department of Computer Science,
Str Targu din Vale 1, 110040 Pitesti, Romania
florentin.ipate@ifsoft.ro

This paper presents a basic framework to define testing strategies for some classes of P systems. Techniques based on grammars and finite state machines are developed and some testing criteria are identified and illustrated through simple examples.

1 Introduction

In 1998, Gheorghe Păun initiated the field of research called *membrane computing* with a paper firstly available on the web [16]. Membrane computing, a new computational paradigm, aims at defining computational models which are inspired by the functioning and structure of the living cell. In particular, membrane computing starts from the observation that compartmentalisation through membranes is one of the essential features of (eucaryotic) cells. Unlike bacterium, which generally consists of a single intracellular compartment, an eucaryotic cell is sub-divided into distinct compartments with well-defined structures and functions. Further on have been considered other biological phenomena like tissues, colonies of different organisms, various bio-chemical entities with dynamic structure in time and space. Membrane systems, also called *P systems*, consist now of different computational models addressing multiple levels of bio-complexity. There are *cell-like P systems*, relying on the hierarchical structure of the living cells, *tissue-like models*, reflecting the network structure of neurons and other bio-units arranged in tissues or more complex organs, *P colonies* and *population P systems*, drawing inspiration from the organisation and behaviour of bacterium colonies, social insects and other organisms living together in larger communities (see [18], [19]).

The most basic model and the first introduced, [17], the cell-like paradigm has three essential features: (i) a *membrane structure* consisting of a hierarchical arrangement of several compartments, called *regions*, delimited by *membranes*; (ii) *objects* occurring inside these regions, coding for various simple or more complex chemical molecules

or compounds; and (iii) *rules* assigned to the regions of the membrane structure, acting upon the objects inside. In particular, each region is supposed to contain a finite set of rules and a finite multiset (or set) of objects. Rules encode for generic transformation processes involving objects and for transporting them, through membranes, from one region to an adjacent one. The application of the rules is performed in a non-deterministic maximally parallel manner: all the applicable rules that can be used to modify or transport existing objects, must be applied, and this is done in parallel for all membranes. This process abstracts the inherent parallelism that occurs at the cellular level.

Since this model has been introduced for the first time in 1998, many variants of membrane systems have been proposed, a research monograph [18] has been published and regular collective volumes are annually edited – a comprehensive bibliography of P systems can be found at [19]. The most investigated membrane computing topics are related to the computational power of different variants, their capabilities to solve hard problems, like NP-complete ones, decidability, complexity aspects and hierarchies of classes of languages produced by these devices. In the last years there have been significant developments in using the P systems paradigm to model, simulate and formally verify various systems [7]. For some of these applications suitable classes of P systems have been associated with and software packages developed. For these models corresponding formal semantics [1] and verification mechanisms [2] have been produced.

There are well-established application areas where the software specifications developed are also delivered together with a model and associated formal verification procedures. All software applications, irrespective of their use and purpose, are tested before being released, installed and used. Testing is not a replacement for a formal verification procedure, when the former is also present, but a necessary mechanism to increase the confidence in software correctness and ultimately a well-known and very well-established stage in any software engineering project [10]. Although formal verification has been applied for different models based on P systems, testing has been completely neglected so far in this context. In this paper we suggest some initial steps towards building a testing framework and its underpinning theory, based on formal grammars and finite state machines, that is associated to software applications derived from P systems specifications. We develop this testing theory based on formal grammars and finite state machines as they are the closest formalisms to P systems and the testing mechanisms for them are well-investigated. Of course other testing approaches can be considered in this context as well, but all of them require a bigger effort of translation and inevitably difficulties in checking the correctnes of this process and in mapping it back to P systems.

The paper is organised as follows: in section 2 there are introduced basic concepts and definitions; a testing framework based on context-free grammars and finite state machines is built and some examples presented in sections 3 and 4, respectively; conclusions are drawn in section 5.

2 Basic Concepts and Notations

For an alphabet $V = \{a_1, ..., a_p\}$, V^* denotes the set of all strings over V. λ denotes the empty string. For a string $u \in V^*$, $|u|_{a_i}$ denotes the number of a_i occurrences in u. For a string u we associate a vector of non-negative integer values $(|u|_{a_1}, ..., |u|_{a_p})$. We denote this by $\Psi_V(u)$.

A basic cell-like P system is defined as a hierarchical arrangement of membranes identifying corresponding regions of the system. With each region there are associated a finite multiset of objects and a finite set of rules; both may be empty. A multiset is either denoted by a string $u \in V^*$, where the order is not considered, or by $\Psi_V(u)$. The following definition refers to one of the many variants of P systems, namely cell-like P system, which uses non-cooperative transformation and communication rules [18]. We will call these processing rules. Since now onwards we will call this model simply P system.

Definition 1 *A* P system *is a tuple* $\Pi = (V, \mu, w_1, ..., w_n, R_1, ..., R_n)$*, where*

- V *is a finite set, called* alphabet*;*
- μ *defines the membrane structure; a hierarchical arrangement of* n *compartments called* regions *delimited by* membranes*; these membranes and regions are identified by integers 1 to* n*;*
- w_i, $1 \leq i \leq n$, *represents the initial multiset occurring in region* i*;*
- R_i, $1 \leq i \leq n$, *denotes the set of rules applied in region* i*.*

The rules in each region have the form $a \rightarrow (a_1, t_1)...(a_m, t_m)$, where $a, a_i \in V$, $t_i \in \{in, out, here\}$, $1 \leq i \leq m$. When such a rule is applied to a symbol a in the current region, the symbol a is replaced by the symbols a_i with $t_i = here$; symbols a_i with $t_i = out$ are sent to the outer region and symbols a_i with $t_i = in$ are sent into one of the regions contained in the current one, arbitrarily chosen. In the following definitions and examples all the symbols $(a_i, here)$ are used as a_i. The rules are applied in maximally parallel mode which means that they are used in all the regions in the same time and in each region all symbols that may be processed, must be.

A configuration of the P system Π is a tuple $c = (u_1, ..., u_n)$, $u_i \in V^*$, $1 \leq i \leq n$. A derivation of a configuration c_1 to c_2 using the maximal parallelism mode is denoted by $c_1 \Longrightarrow c_2$. In the set of all configurations we will distinguish terminal configurations; $c = (u_1, ..., u_n)$ is a terminal configuration if there is no region i such that u_i can be further processed.

The set of all halting configurations is denoted by $L(\Pi)$, whereas the set of all configurations reachable from the initial one (including the initial configuration) is denoted by $S(\Pi)$.

Definition 2 *A* deterministic finite automaton *(abbreviated* DFA*),* M*, is a tuple* (A, Q, q_0, F, h)*, where:*

- A *is the finite* input alphabet*;*
- Q *is the finite* set of states*;*
- $q_0 \in Q$ *is the* initial state*;*
- $F \subseteq Q$ *is the* set of final states*;*
- $h : Q \times A \longrightarrow Q$ *is the* next-state *function.*

The next-state function h can be extended to a function $h : Q \times A^* \longrightarrow Q$ defined by:

- $h(q, \epsilon) = q, q \in Q$;
- $h(q, sa) = h(h(q, s), a), q \in Q, s \in A^*, a \in A$.

For simplicity the same name h is used for the next-state function and for the extended function.

Given $q \in Q$, a sequence of input symbols $s \in A^*$ is said to be accepted by M in q if $h(q, s) \in F$. The set of all input sequences accepted by M in q_0 is called the *language defined (accepted) by* M, denoted $L(M)$.

3 Grammar-like Testing

Based on testing principles developed for context-free grammars [13], [14], some testing strategies aiming to achieve rule coverage for a P system will be defined and analysed in this section.

In *grammar engineering*, formal grammars are used to specify complex software systems, like compilers, debuggers, documentation tools, code pre-processing tools etc. One of the areas of grammar engineering is *grammar testing* which covers the development of various testing strategies for software based on grammar specifications. One of the main testing methods developed in this context refers to rule coverage, i.e., the testing procedure tries to cover all the rules of a specification.

In the context of grammar testing it is assumed that for a given specification defined as a grammar, an implementation of it, like a parser, exists and will be tested. The aim is to build a test set, a finite set of sequences, that reveals potential errors in the implementation. As opposed to testing based on finite state machines, where it is possible to (dis)prove the equivalent behaviour of the specification and implementation, in the case of general context-free grammars this is no longer possible as it reduces to the equivalence of two such devices, which is not decidable. Of course, for specific restricted

classes of context-free grammars there are decidability procedures regarding the equivalence problem and these may be considered for testing purposes as well, but we are interested here in the general case. The best we can get is to cover as much as possible from the languages associated to the two mechanisms, specification and implementation grammars, and this is the role of a test set. We will define such test sets for P systems.

Given a specification G and an implementation G', a test set aims to reveal inconsistencies, like

- *incorrectness* of the implementation G' with respect to the specification language $L = L(G)$, if $L(G') \not\subset L$ and $L \neq L(G')$;
- *incompleteness* of the implementation G' with respect to the specification language $L = L(G)$, if $L \not\subset L(G')$ and $L \neq L(G')$.

We start to develop a similar method in the context of P systems. Although there are a number of similarities between context-free grammars utilised in grammar testing and basic P systems, like those considered in this paper, there are also major differences that pose new problems in defining suitable testing methods. Some of the difficulties that we encounter in introducing some grammar-like testing procedures are related to the main features that define such systems: hierarchical compartmentalisation of the entire model, parallel behaviour, communication mechanisms, the lack of a non-terminal alphabet.

We define some rule coverage criteria by firstly starting with one compartment P system, i.e., $\Pi = (V, \mu, w, R)$, where $\mu = [_1]_1$. The rule coverage criteria are adapted from [13], [14]. In the sequel, if not otherwise stated, we will consider that the specification and the implementation are given by the P systems Π and Π', respectively. For such a P system Π, we define the following concepts.

Definition 3 *A multiset denoted by $u \in V^*$,* covers *a rule $r : a \to v \in R$, if there is a derivation $w \Longrightarrow^* xay \Longrightarrow x'vy' \Longrightarrow^* u$; $w, x, y, v, u \in V^*$, $a \in V$.*

If there is no further derivation from u, then this is called a *terminal coverage*.

Definition 4 *A set $T \subseteq V^*$, is called a* test set *that satisfies the* rule coverage *(RC) criterion if for each rule $r \in R$ there is $u \in T$ which covers r.*

If every $u \in T$ provides a terminal coverage then T is called a test set that satisfies the *rule terminal coverage* (RTC) criterion.

The following one compartment P systems are considered, $\Pi_i, 1 \leq i \leq 4$, having the same alphabet and initial multiset:

$$\Pi_i = (V_i, \mu_i, w_i, R_i)$$

where

- $V_1 = V_2 = V_3 = V_4 = \{s, a, b, c\}$;
- $\mu_1 = \mu_2 = \mu_3 = \mu_4 = [_1]_1$ - i.e., one compartment, denoted by 1;
- $w_1 = w_2 = w_3 = w_4 = s$;
- $R_1 = \{r_1 : s \to ab, r_2 : a \to c, r_3 : b \to bc, r_4 : b \to c\}$;
- $R_2 = \{r_1 : s \to ab, r_2 : a \to \lambda, r_3 : b \to c\}$;
- $R_3 = \{r_1 : s \to ab, r_2 : a \to bcc, r_3 : b \to \lambda\}$;
- $R_4 = \{r_1 : s \to ab, r_2 : a \to bc, r_3 : a \to c, r_4 : b \to c\}$.

In the sequel for each multiset w, we will use the following vector of non-negative integer numbers $(|w|_s, |w|_a, |w|_b, |w|_c)$.

The sets of all configurations expressed as vectors of non-negative integer numbers, computed by the P systems Π_i, $1 \leq i \leq 4$ are:

- $S(\Pi_1) = \{(1,0,0,0), (0,1,1,0)\} \cup \{(0,0,k,n) | k = 0,1; n \geq 2\}$;
- $S(\Pi_2) = \{(1,0,0,0), (0,1,1,0), (0,0,0,1)\}$;
- $S(\Pi_3) = \{(1,0,0,0), (0,1,1,0), (0,0,1,2), (0,0,0,2)\}$;
- $S(\Pi_4) = \{(1,0,0,0), (0,1,1,0), (0,0,1,2), (0,0,0,2), (0,0,0,3)\}$.

Test sets for Π_1 satisfying the RC criterion are

- $T_{1,1} = \{(0,1,1,0), (0,0,1,2), (0,0,0,2)\}$ and
- $T_{1,2} = \{(0,1,1,0), (0,0,1,2), (0,0,0,3)\}$,

whereas $T'_{1,1} = \{(0,1,1,0), (0,0,0,2)\}$ and $T'_{1,2} = \{(0,1,1,0), (0,0,1,2)\}$ are not, as they do not cover the rules r_3 and r_4, respectively. Both $T_{1,1}$ and $T_{1,2}$ show the incompleteness of Π_2 with respect to $S(\Pi_1)$ (Π_2 is also incorrect). $T_{1,1}$ does not show the incompleteness of Π_3 with respect to $S(\Pi_1)$, but $T_{1,2}$ does. None of these test sets does show the incompleteness of Π_4.

The sets of terminal configurations expressed as vectors of non-negative integer numbers, computed by the P systems Π_i, $1 \leq i \leq 4$ are:

- $L(\Pi_1) = \{(0,0,0,n) | n \geq 2\}$;
- $L(\Pi_2) = \{(0,0,0,1)\}$;
- $L(\Pi_3) = \{(0,0,0,2)\}$;
- $L(\Pi_4) = \{(0,0,0,2), (0,0,0,3)\}$.

A test set for Π_1 satisfying the RTC criterion is $T = \{(0,0,0,3)\}$. As $(0,0,0,3)$ is not in $L(\Pi_2)$ and $L(\Pi_3)$, it follows that Π_2 and Π_3 are incomplete with respect to $L = L(\Pi_1)$. However, the test set does not prove the incompleteness of Π_4.

The examples above show that none of the test sets provided is powerful enough to prove the incompleteness of Π_4, although $S(\Pi_4) \subset S(\Pi_1)$, and $L(\Pi_4) \subset L(\Pi_1)$.

A more powerful testing set is computed by considering a generalisation of the RC criterion, called *context-dependent rule coverage* (CDRC) criterion.

Definition 5 *A rule $r \in R$, $r : b \to uav$, $u, v \in V^*$, $a, b \in V$, is called a* direct occurrence *of a. For every symbol $a \in V$, we denote by $Occs(\Pi, a)$, the set of all direct occurrences of a.*

For the P system Π_1, the following sets of direct occurrences are computed:

- $Occs(\Pi_1, s) = \emptyset$;
- $Occs(\Pi_1, a) = \{r_1 : s \to ab\}$;
- $Occs(\Pi_1, b) = \{r_1 : s \to ab, r_3 : b \to bc\}$;
- $Occs(\Pi_1, c) = \{r_2 : a \to c, r_3 : b \to bc, r_4 : b \to c\}$.

Definition 6 *A multiset denoted by $u \in L(\Pi)$* covers *the rule $r : b \to y \in R$ for the* direct occurrence *of b, $a \to ubv \in R$ if there is a derivation $w \Longrightarrow^* u_1 a v_1 \Longrightarrow u_1 ubv v_1 \Longrightarrow u_1 uyvv_1 \Longrightarrow^* z$; $z, u_1, v_1, u, v, y \in V^*$, $a, b \in V$. A set T_r is said to cover $r : a \to x$ for all direct occurrences of a if for any occurrence $o \in Occs(\Pi, a)$ there is $t \in T_r$ such that t covers r for o. A set T is said to achieve CDRC for Π if it covers all $r \in R$ for all their direct occurrences.*

Clearly, T_r covers r in the sense of Definition 3. Similar to the coverage rule criterion introduced by Definition 4 where a terminal coverage criterion (RTC) has been given, we can also extend CDRC by considering only terminal derivations for all z in Definition 6 and obtain the CDRTC criterion. Obviously, any test set that satisfies CDRC (CDRTC) criterion will also satisfies RC (RTC) criterion, as well.

For Π_1 the set

- $T' = \{(0,1,1,0), (0,0,1,2), (0,0,0,2), (0,0,1,3), (0,0,0,3))\}$ satisfies the CDRC criterion and
- $T'' = \{(0,0,0,2), (0,0,0,3), (0,0,0,4)\}$ satisfies the CDRTC criterion.

These sets show the incompleteness of Π_4 as well as the incompleteness of the other two P systems.

In all the above considerations we have considered maximal parallelism. If we consider sequential P systems, only one rule is used at a moment, then all the above considerations and the same sets remain valid.

Now we consider general P systems, as introduced by Definition 1, and reformulate the concepts introduced above for one compartment P systems:

- RC criterion becomes: the configuration $(u_1, ..., u_i, ..., u_n)$ covers a rule $r_i : a_i \to v_i \in R_i$ if there is a derivation
  $(w_1, ..., w_i..., w_n) \Longrightarrow^* (x_1, ..., x_i a_i y_i, ..., x_n) \Longrightarrow (x'_1, ..., x'_i v_i y'_i, ..., x'_n) \Longrightarrow^* (u_1, ..., u_i, ..., u_n)$;
- a test set $T \subseteq (V^*)^n$ is defined similar to Definition 4.

In a P system with more than one compartment, two adjacent regions can exchange symbols. If the region i is contained in j and $r_i : a \to x(b, out)y \in R_i$ or $r_j : c \to u(d, in)v \in R_j$ then r_i, r_j are called communication rules between regions i and j.

Now Definition 5 can be rewritten as follows

Definition 7 *A rule $r : b \to uav \in R_i$ or a communication rule between i and j, $r' : b' \to u'(a, t)v' \in R_j$, where i and j are two adjacent regions and $t \in \{in, out\}$, is called a direct occurrence of a. The set of all direct occurrences of a in region i is denoted by $Occs_i(\Pi, a)$ and consists of the set of all direct occurrences of a from i, denoted by S_i and the sets of communication rules, r', from the adjacent regions $j_1, ..., j_q$, denoted by $S_{j_1}, ..., S_{j_q}$.*

Let the two compartment P systems:

$$\Pi'_i = (V_i, \mu_i, w_{1,i}, w_{2,i}, R_{1,i}, R_{2,i})$$

where

- $V_1 = V_2 = \{s, a, b, c\}$;
- $\mu_1 = \mu_2 = [_1[_2]_2]_1$ - i.e., two compartments; region 1 contains 2;
- $w_{1,1} = s, w_{2,1} = \lambda; w_{1,2} = s, w_{2,2} = \lambda$;
- $R_{1,1} = \{r_1 : s \to sa(b, in), r_2 : s \to ab, r_3 : b \to a, r_4 : a \to c\}$;
- $R_{2,1} = \{r_1 : b \to bc, r_2 : b \to c\}$;
- $R_{1,2} = \{r_1 : s \to sa(b, in), r_2 : s \to ab(b, in)(c, in), r_3 : b \to a, r_4 : a \to c\}$;
- $R_{2,2} = \{r_1 : b \to \lambda, r_2 : b \to c\}$.

For the P system Π'_1, the following sets of direct occurrences are computed:

- $Occs_1(\Pi_1, a) = S_1 \cup S_2$, where $S_1 = \{r_1 : s \to sa(b, out), r_2 : s \to ab, r_3 : b \to a\}$ and $S_2 = \emptyset$;

- $Occs_2(\Pi'_1, b) = S_1 \cup S_2$, where $S_1 = \{r_1 : s \rightarrow sa(b, out)\}$ and $S_2 = \{r_1 : b \rightarrow bc\}$.

A test set T that satisfies the CDRC criterion is:

$$\{((1,1,0,0),(0,0,1,0)),((0,1,1,1),(0,0,1,1)),((0,1,0,2),(0,0,0,2)),$$
$$((0,0,0,3),(0,0,0,2))\},$$

which is obtained from the derivation

$$(s, \lambda) \Longrightarrow (sa, b) \Longrightarrow (bac, bc) \Longrightarrow (acc, cc) \Longrightarrow (ccc, cc).$$

The P system Π'_2 is incomplete as it does not contain configurations (c^k, c^h) with $h > k$, but T above, fails to show this fact.

We can consider the CDRTC criterion to check whether it distinguishes between Π'_1 and Π'_2. It is left to the reader to verify this fact.

4 Finite State Machine based Testing

In this section we apply concepts and techniques from finite state based testing. In order to do this, we construct a finite automaton on the basis of the derivation tree of a P system.

We first present the process of constructing a DFA for a one compartment P system $\Pi = (V, \mu, w, R)$, where $\mu = [_1]_1$. In this case, the configuration of Π can change as a result of the application of some rule in R or of a number of rules, in parallel. In order to guarantee the finiteness of this process, for a given integer k, only computations of maximum k steps will be considered. For example, for $k = 4$, the tree in Figure 4.1 depicts all derivations in Π_1 of length less than or equal to k. The terminal nodes are in bold.

As only sequences of maximum k steps are considered, for every rule $r_i \in R$ there will be some N_i such that, in any step, r_i can be applied at most N_i times. Thus, the tree that depicts all the derivations of a P system Π with rules $R = \{r_1, \ldots, r_m\}$ can be described by a DFA Dt over the alphabet $A = \{r_1^{i_1} \ldots r_m^{i_m} \mid 0 \leq i_1 \leq N_1, \ldots, 0 \leq i_m \leq N_m\}$, where $r_1^{i_1} \ldots r_m^{i_m}$ describes the multiset with i_j occurrences of r_j, $1 \leq j \leq m$.

As Dt is a DFA over A, one can construct the minimal DFA that accepts *precisely* the language $L(Dt)$ defined by Dt. However, as only sequences of at most k transitions are considered, it is irrelevant how the constructed automaton will behave for longer sequences. Thus, a finite cover automaton can be constructed instead.

A *deterministic finite cover automaton* (*DFCA*) of a finite language U is a DFA that accepts all sequences in U and possibly other sequences that are longer than any sequence in U.

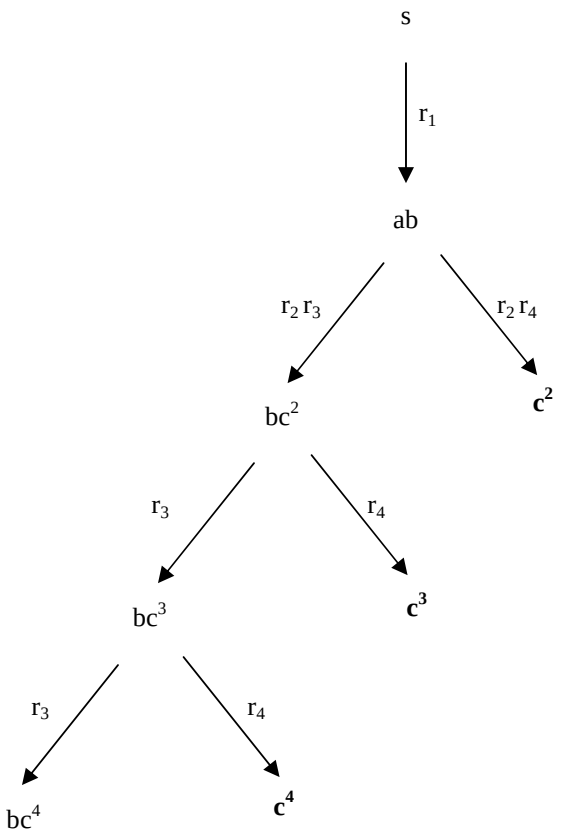


Fig. 4.1 Derivation tree for Π_1 and $k = 4$

Definition 8 *Let $M = (A, Q, q_0, F, h)$ be a DFA, $U \subseteq A^*$ a finite language and l the length of the longest sequence(s) in U. Then M is called a* deterministic finite cover automaton *(*DFCA*) of U if $L(A) \cap A[l] = U$, where $A[l] = \bigcup_{0 \leq i \leq l} U^i$ denotes the sets of sequences of length less than or equal to l with members in the alphabet A.*

A *minimal* DFCA of U is a DFCA of U having the least number of states. Unlike the case in which the acceptance of the precise language is required, the minimal DFCA is not necessarily unique (up to a renaming of the state space) [5], [6].

The concept of DFCA was introduced by Câmpeanu et al. [5], [6] and several algorithms for constructing a minimal DFCA of a finite language have been devised since [5], [6], [4], [3], [11], [12], [16]. The time complexity of these algorithms is polynomial in the number of states of the minimal DFCA. Interestingly, García and Ruiz [8] note that the minimization of DFCA can be approached as an inference problem, which had been solved several years earlier.

Any DFA that accepts U is also a DFCA of U and so the size (number of states) of a minimal DFCA of U cannot exceed the size of the minimal DFA that accepts U. On the other hand, as shown by examples in this paper, a minimal DFCA of U may have considerably fewer states than the minimal DFA that accepts U.

A minimal DFCA of the language $L(Dt)$ defined by the previous derivation tree is represented in Figure 4.2; q_3 in Figure 4.2 is final state. It is implicitly assumed that a non-final "sink" state, denoted q_S, also exists, that receives all "rejected" transitions. For testing purposes we will consider all the states as final.

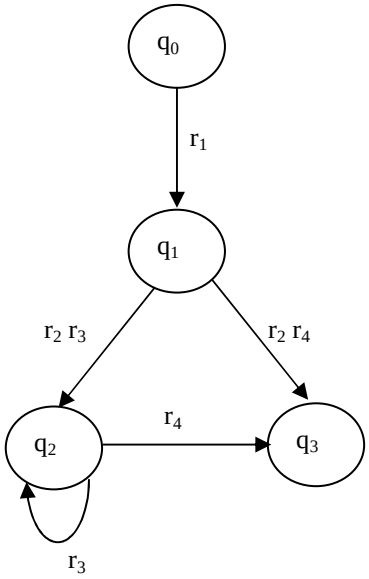


Fig. 4.2 Minimal DFCA for Π_1 and $k = 4$

Not only the minimal DFCA of $L(Dt)$ may have (significantly) less states than the minimal DFA that accepts $L(Dt)$, but it also provides the right approximation for the computation of a P system. Consider $u_1, u_2 \in V^*$, $w \Longrightarrow^* u_1$, $w \Longrightarrow^* u_2$, such that the derivation from u_1 is identical to the derivation from u_2, i.e., any sequence $s \in A^*$ that can be applied to u_1 can also be applied to u_2 and vice versa (e.g., $u_1 = bc^2$ and $u_2 = bc^3$ in Figure 4.1). Naturally, as the derivation from u_1 is identical to the derivation from u_2, u_1 and u_2 should be represented by the same state of a DFA that models the computation of the P system. This is the case when the DFA model considered is a minimal DFCA of $L(Dt)$; on the other hand, u_1 and u_2 will be associated with distinct states in the minimal DFA that accepts $L(Dt)$, unless they appear at the same level in the derivation tree Dt. For example, in Figure 4.1, bc^2 and bc^3 appear at different levels in the derivation tree and so they will be associated with distinct states in the minimal DFA that accepts $L(Dt)$; on the other hand, bc^2 and bc^3 are mapped onto the same state (q_2)

of the minimal DFCA represented in Figure 4.2. Furthermore, if the entire computation of the P system (i.e. for derivation sequences of *any* length) can be described by a DFA over some alphabet A, then this DFA model will be obtained as a DFCA of $L(Dt)$ for k sufficiently large.

Once the minimal DFCA $M = (A, Q, q_0, F, h)$ has been constructed, various specific coverage levels can be used to measure the effectiveness of a test set. In this paper we use two of the most widely known coverage levels for finite automata: *state coverage* and *transition coverage.*

Definition 9 *A set $T \subseteq V^*$, is called a* test set *that satisfies the* state coverage *(SC) criterion if for each state q of M there exists $u \in T$ and a path $s \in A^*$ that reaches q ($h(q_0, s) = q$) such that u is derived from w through the computation defined by s.*

Definition 10 *A set $T \subseteq V^*$, is called a* test set *that satisfies the* transition coverage *(TC) criterion if for each state q of M and each $a \in A$ such that a labels a valid transition from q ($h(q, a) \neq q_S$), there exist $u, u' \in T$ and a path $s \in A^*$ that reaches q such that u and u' are derived from w through the computation defined by s and sa, respectively.*

Clearly, if a test set satisfies TC, it also satisfies SC. A test set for Π_1 satisfying the SC criterion is
$T_{1,1} = \{(1,0,0,0), (0,1,1,0), (0,0,1,2), (0,0,0,2)\}$,
whereas a test set satisfying the TC criterion is
$T_{1,s} = \{(1,0,0,0), (0,1,1,0), (0,0,1,2), (0,0,0,2), (0,0,1,3), (0,0,0,3)\}$.

The TC coverage criterion defined above is, in principle, analogous to the RC criterion given in the previous section. The TC criterion, however, does not only depend on the rules applied, but also on the state reached by the system when a given rule has been applied. Test suites that meet the RC and TC criteria can be efficiently calculated using automata inference techniques [9], [15]. A stronger criterion, in which all feasible transition sequences of length less than or equal to 2 must be triggered in any state can also be defined - this will correspond to the CDRC criterion defined in the previous section. Of course, a more careful analysis of the relationships between criteria used in testing based on grammars and those applied in the context of finite state machines, considered for P systems testing, needs to be conducted in order to identify the most suitable testing procedures for these systems.

The construction of a minimal DFCA and the coverage criteria defined above can be generalized for a multiple compartment P system

$$\Pi = (V, \mu, w_1, ..., w_n, R_1, ..., R_n).$$

In this case, the input alphabet will be defined as

$$A = \{(r_1^{i_{1,1}} \dots r_{m_1}^{i_{1,m_1}}, \dots, r_1^{i_{n,1}} \dots r_{m_n}^{i_{n,m_n}}) \mid 0 \leq i_{j,p} \leq N_{j,p}, 1 \leq j \leq m_p,$$

$$1 \leq p \leq n\},$$

where $N_{j,p}$ is the maximum number of times rule r_j, $1 \leq j \leq m_p$ from compartment p can be applied in one derivation step, $1 \leq p \leq n$. Analogously to one compartment P systems, only computations of maximum k steps are considered.

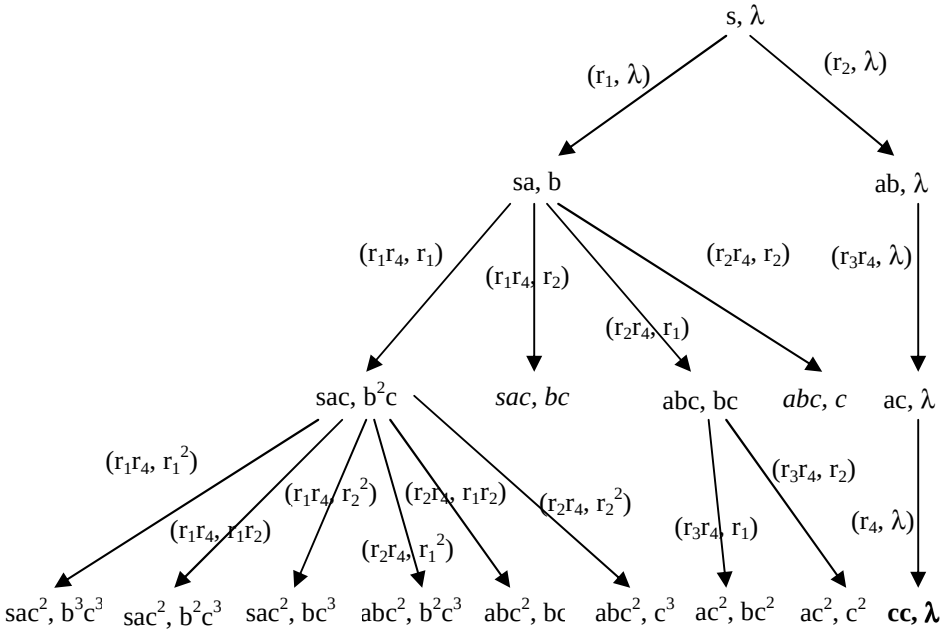


Fig. 4.3 Derivation tree for Π_1' and $k = 3$

For $k = 3$, the derivation tree of Π_1' defined above is as represented in Figure 4.3. For clarity, in Figure 4.3 if the derivation from some node u (not found at the bottom level in the hierarchy) is the same as the derivation from some previous node u' at a higher level or at the same level in the hierarchy, then u is not expanded any further; we denote $u \sim u'$. Such nodes are (sac, bc) and (abc, c), $(sac, bc) \sim (sa, b)$ and $(abc, c) \sim (ab, \lambda)$; they are given in italics. A minimal DFCA of the language defined by the derivation tree is represented in Figure 4.4.

Similar to one compartment case, test sets for considered criteria, state and transition cover, can be defined in this more general context.

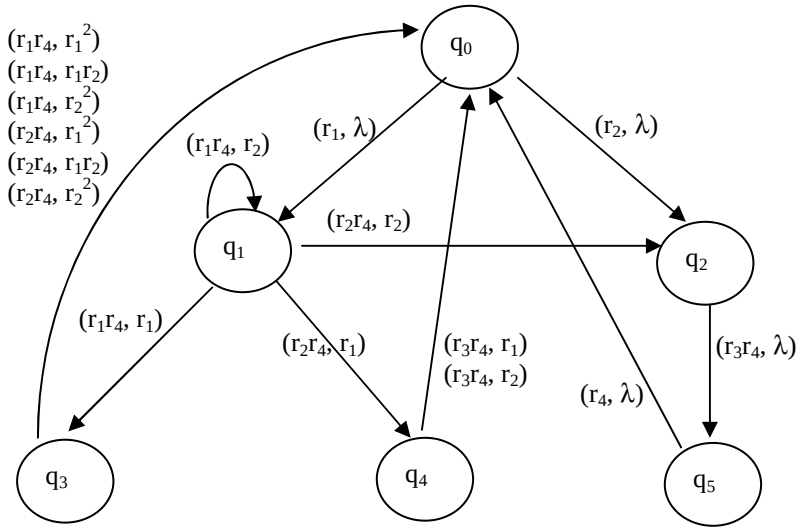


Fig. 4.4 Minimal DFCA for Π'_1 and $k = 3$

5 Conclusions and Future Work

In this paper we have discussed how P systems are tested by introducing grammar and finite state machine based strategies. The approach is focussing on cell-like P systems, but the same methodology can be used for tissue-like P systems. Simple examples illustrate the approach and show their testing power as well as current limitations. This very initial research reveal a number of interesting preliminary problems regarding the construction of relevant test sets that point out faulty implementations.

This paper has focused on coverage criteria for P system testing. In grammar based testing, coverage is the most widely used test generation criteria. For finite state based testing we have considered some simple state and transition coverage criteria, but criteria for conformance testing (based on equivalence proofs) can also be used; this approach is the subject of a paper in progress. Future research is also intended to cover relationships between components and the whole systems with respect to testing, other testing principles based on the same criteria and strategies, as well as new strategies and different testing methods. Relationships between testing criteria based on grammars and those used in the case of finite state machine based specifications remain to be further investigated in a more general context.

Acknowledgements. The authors of the paper are grateful to the anonymous referees for their comments and suggestions.

Bibliography

[1] O. Andrei, G. Ciobanu, D. Lucanu (2005). Structural operational semantics of P systems. In *WMC6, LNCS* **3850**, 31–48.

[2] O. Andrei, G. Ciobanu, D. Lucanu (2007). A rewriting logic framework for operational semantics of membrane systems. *Theoretical Computer Science*, **373**, (3), 163–181.

[3] C. Câmpeanu, A. Păun, J. R. Smith (2006). Incremental construction of minimal deterministic finite cover automata. *Theoretical Computer Science*, **363**, (2), 135–148.

[4] C. Câmpeanu, A. Păun, S. Yu (2002). An efficient algorithm for constructing minimal cover automata for finite languages. *International Journal of Foundation of Computer Science*, **13**, (1), 83–97.

[5] C. Câmpeanu, N. Santean, S. Yu (1998). Minimal cover-automata for finite languages. In *Workshop on Implementing Automata, LNCS* **1660**, 43–56.

[6] C. Câmpeanu, N. Santean, S. Yu (2001). Minimal cover-automata for finite languages. *Theoretical Computer Science*, **267**, (1-2), 3–16.

[7] G. Ciobanu, Gh. Păun, M. J. Pérez-Jiménez, eds. (2006). *Applications of membrane computing*, Springer, Berlin.

[8] P. García, Jose Ruiz (2004) A note on the minimal cover-automata for finite languages", *Bulletin of the EATCS* **83**, 193–194.

[9] M. E. Gold (1978) Complexity of Automaton Identification from Given Data, *Information and Control*, **37**, 302–320.

[10] M. Holcombe, F. Ipate (1998). *Correct systems - Building business process solutions*, Springer, Berlin.

[11] H. Körner (2002). On minimizing cover automata for finite languages in O(n log n) time. In *CIAA, LNCS*, **2608** 117–127.

[12] H. Körner (2003). A time and space efficient algorithm for minimizing cover automata for finite languages. *International Journal of Foundation of Computer Science*, **14**, (6), 1071–1086.

[13] R. Lämmel (2001). Grammar testing, *FASE, LNCS*, **2029**, 201–216.

[14] H. Li, M. Jin, C. Liu, Z. Gao (2004). Test criteria for context-free grammars, *COMPSAC*, Vol. **1**, 300–305.

[15] J. Oncina, P. Garca (1992) Inferring regular languages in polynomial update time, in, *N. Prez de la Blanca, A. Sanfeliu and E.Vidal (Eds.), Pattern recognition and image analysis, vol. 1 of Series in Machine Perception and Artificial Intelligence*, World Scientific, 49-61.

[16] A. Păun, N. Santean, S. Yu (2000). An O(n^2) algorithm for constructing minimal cover automata for finite languages. In *CIAA, LNCA*, **2088** 243–251.

[17] Gh. Păun (2000). Computing with membranes, *Journal of Computer and System Sciences*, **61**, (1), 108 – 143.

[18] Gh. Păun (2002). *Membrane Computing. An introduction*, Springer, Berlin.

[19] The P Systems Web Site: `http://ppage.psystems.eu`

A Spiking Neural P System Based Model for Hebbian Learning

Miguel A. Gutiérrez-Naranjo and Mario J. Pérez-Jiménez

University of Sevilla, Department of Computer Science and Artificial Intelligence,
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
{magutier,marper}@us.es

Spiking neural P systems and *artificial neural networks* are computational devices which share a biological inspiration based on the flow of information among neurons. In this paper we present a first model for Hebbian learning in the framework of Spiking Neural P systems by using concepts borrowed from neuroscience and artificial neural network theory.

1 Introduction

> *When an axon of cell A is near enough to excite cell B or repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.*
>
> *D. O. Hebb (1949) [14]*

Neuroscience has been a fruitful research area since the pioneering work of Ramón y Cajal in 1909 [23] and after a century full of results on the man and the mind, many interesting questions are still unanswered. Two of such problems of current neuroscience are the understanding of neural plasticity and the neural coding.

The first one, the understanding of neural plasticity, is related to the changes in the amplitude of the postsynaptic response to an incoming action potential. Electrophysiological experiments show that the response amplitude is not fixed over time. Since the 1970's a large body of experimental results on synaptic plasticity has been accumulated. Many of these experiments are inspired by Hebb's postulate (see above). In the integrate-and-fire formal spiking neuron model [9] and also in artificial neural networks [13], it is usual to consider a factor w as a measure of the *efficacy* of the synapse from a neuron to another.

The second one, the neural coding, is related to the way in which one neuron sends information to other ones. It is interested in the information contained in the spatio-temporal pattern of pulses and on the code used by the neurons to transmit information.

This research area wonders how other neurons decode the signal or whether the code can be read by external observers and understand the message. At present, a definite answer to these questions is not known.

The elementary processing units in the central nervous system are neurons which are connected to each other in an intricate pattern. Cortical neurons and their connections are packed into a dense network with more than 10^4 cell bodies per cubic millimeter. A single neuron in a vertebrate cortex often connects to more than 10^4 postsynaptic neurons.

The neuronal signals consist of short electrical pulses (also called action potentials or *spikes*) and can be observed by placing a fine electrode close to the soma or axon of a neuron. The link between two neurons is a *synapse* and it is common to refer to the sending neuron as a presynaptic cell and to the receiving neuron as the postsynaptic cell.

Since all spikes of a given neuron look alike, the form of the action potential does not carry any information. Rather, it is the number and the timing of spikes what matters. Traditionally, it has been thought that most, if not all, of the relevant information was contained in the *mean* firing rate of the neuron. The concept of mean firing rates has been successfully applied during the last decades (see, e.g., [19] or [15]) from the pioneering work of Adrian [1, 2]. Nonetheless, more and more experimental evidence has been accumulated during recent years which suggests that a straightforward firing rate concept based on temporal averaging may be too simplistic to describe brain activity. One of the main arguments is that reaction times in behavioral experiment are often too short to allow long temporal averages. Humans can recognize and respond to visual scenes in less than 400ms [25]. Recognition and reaction involve several processing step from the retinal input to the finger movement at the output. If at each processing steps, neurons had to wait and perform a temporal average in order to read the message of the presynaptic neurons, the reaction time would be much longer. Many other studies show the evidence of precise temporal correlations between pulses of different neurons and stimulus-dependent synchronization of the activity in populations of neurons (see, for example, [5, 11, 10, 6, 24]). Most of these data are inconsistent with a concept of coding by mean firing rates where the exact timing of spikes should play no role.

Instead of considering mean firing rates, we consider the realistic situation in which a neuron abruptly receives an input and for each neuron the timing of the first spike after the reference signal contains all the information about the new stimulus.

Spiking neural P systems (SN P systems, for short) were introduced in [16] with the aim of incorporating in membrane computing[1] ideas specific to spike-based neuron models. The intuitive goal was to have a directed graph were the nodes represent the neurons and the edges represent the synaptic connections among the neurons. The flow of information is carried on the action potentials, which are encoded by objects of the same

[1] The foundations of membrane computing can be found in [21] and updated bibliography at [26].

type, the *spikes*, which are placed inside the neurons and can be sent from presynaptic to postsynaptic neurons according to specific rules and making use of the time as a support of information.

This paper is a first answer to the question proposed by Gh. Păun in [22] (question A) about the way to link the study of SN P systems with neural computing and as he suggests, the starting point has been not only neural computing, but also recent discoveries in neurology.

The paper is organized as follows: first we discuss about SN P systems with input and delay and a new computational device called Hebbian SN P system unit is presented. In Section 3 we present our model of learning with SN P systems based on Hebb's postulate. An illustrative experiment carried out with the corresponding software is shown in Section 4. Finally, some conclusions and further discussion on some topics of the paper are given in the last section.

2 SN P Systems with Input and Decay

An SN P system consists of a set of neurons placed in the nodes of a directed graph and capable of sending signals (called *spikes*) along the arcs of the graph (called *synapses*) according to specific rules. The objects evolve according to a set of rules (called *spiking rules*). The idea is that a neuron containing a certain amount of spikes can consume some of them and produce other ones. The produced spikes are sent (maybe with a delay of some steps) to all adjacent neurons from the neuron where the rule was applied. A global clock is assumed and in each time unit, each neuron which can use a rule should do it, but only (at most) one rule is used in each neuron. One distinguished neuron is considered to be the output neuron, and its spikes are also sent to the environment (a detailed description of SN P systems can be found in [22] and the references therein).

In this section we introduce the *Hebbian SN P system unit* which is an SN P system with $m + 1$ neurons (m presynaptic neurons linked to one postsynaptic neuron) endowed with *input* and *decay*. At the starting point all the neurons are inactive. At rest, the membrane of biological neurons has a negative polarization of about $-65mV$, but we will consider the inactivity by considering the number of spikes inside the neuron is zero. The dynamics of a Hebbian SN P system unit is quite natural. At the starting point, all neurons are at rest and in a certain moment the presynaptic neurons receive spikes enough to activate some rules. The instant of the arrival of the spikes can be different for each presynaptic neuron. These spikes activate one rule inside the neurons and the presynaptic neurons send spikes to the postsynaptic neuron. In the postsynaptic neuron a new rule can be triggered or not, depending on the arrival of spikes and it may send a spike to the environment.

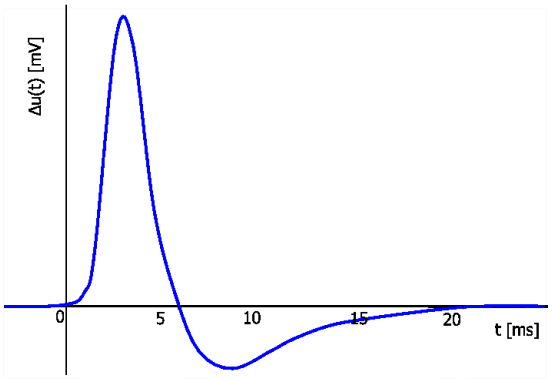


Fig. 2.1 Dynamics of one spike

2.1 The Input The basic idea in SN P systems taken from biological spiking neuron models is that the information is encoded in *time*. The information in a Hebbian SN P system unit is also encoded in the time in which the spikes arrive to the neuron and the time in which the new spikes are emitted. The input will be also encoded in time. The idea behind this codification is that the presynaptic neurons may not be activated at the same moment. If we consider a Hebbian SN P system unit as part of a wide neural network, it is quite natural to think that the spikes will not arrive to the presynaptic neurons (and consequently, their rules are not activated) at the same time. In this way, if we consider a Hebbian SN P system unit with m presynaptic neurons $\{u_1, \ldots, u_m\}$, an input will consist of a vector $\vec{x} = (x_1, \ldots, x_m)$ of non-negative integers where x_i represents the time unit of the global clock in which the neuron u_i is activated[2].

2.2 The Decay The effect of a spike on the postsynaptic neuron can be recorded with an intracellular electrode which measures the potential difference between the interior of the cell and its surroundings. Without any spike input, the neuron is at rest corresponding to a constant membrane potential. After the arrival of the spike, the potential changes and finally decays back to the resting potential. The spikes, have an amplitude of about 100mV and typically a duration of 1-2 ms. This means that if the total change of the potential due to the arrival of spikes is not enough to activate the postsynaptic neuron, it decays after some milliseconds and the neuron comes back to its resting potential (see Fig. 2.1).

This biological fact is not implemented in current SN P systems, where the spikes can be inside the neuron for a long time if they are not consumed by any rule. In the Hebbian SN P system unit, we introduce the decay in the action potential of the neurons. When the impulse sent by a presynaptic neuron arrives to the postsynaptic neuron, if it is not

[2] In Section 5 we discuss about other codings for the input.

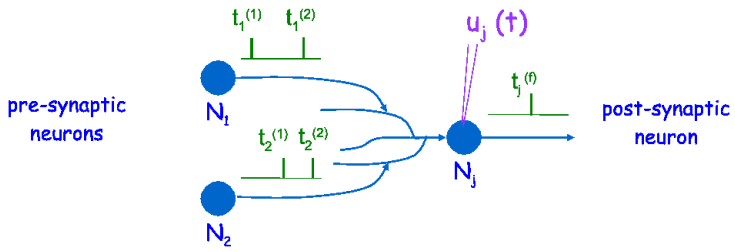


Fig. 2.2 Two presynaptic and one postsynaptic neuron

consumed for triggering any rule in the postsynaptic neuron it decays and its contribution to the total change of potential in the postsynaptic neuron decreases with time. This decayed potential is still able to contribute to the activation of the postsynaptic rule if other spikes arrive to the neuron and the addition of all the spikes trigger any rule. If this one does not occur, the potential decays and after a short time the neuron reaches the potential at rest. Figure 2.2 shows a scheme in which two presynaptic neurons send two spikes each of them at different moments to a postsynaptic neuron. Figure 2.3 shows the changes of potential in the postsynaptic neuron till reaching the threshold for firing a response.

In order to formalize the idea of decay in the framework of SN P systems we introduce a new type of extended rules: the *rules with decay*. They are rules of the form

$$E/a^k \rightarrow (a^p, S); d$$

where, E is a regular expression over $\{a\}$, k and p are natural numbers with $k \geq p \geq 0$, $d \geq 0$ and $S = (s_1, s_2, \ldots, s_r)$ is a finite non-increasing sequence of natural numbers called the *decaying sequence* where $s_1 = k$ and $s_r = 0$. If $E = a^k$, we will write $a^k \rightarrow (a^p, S); d$ instead of $a^k/a^k \rightarrow (a^p, S); d$.

The idea behind the *decaying sequence* is the following. When the rule $E/a^k \rightarrow (a^p, S); d$ is triggered at t_0 we look in $S = (s_1, \ldots, s_r)$ for the least l such that $p \geq s_l$. Such s_l spikes are sent to the postsynaptic neurons according with the delay d in the usual way. Notice that s_l can be equal to p, so at this point this new type of rule is a generalization of the usual extended rules.

This definition of decay[3] can be seen as a generalization of the decaying spikes presented in [7], where a decaying spike a is written in the form (a, e), where $e \geq 1$ is the period. From the moment a pair (a, e) arrives in a neuron, e is decremented by one in each step of computation. As soon as $e = 0$, the corresponding spike is lost and cannot be used anymore.

[3]Further discussion about the decay can be found in Section 5.

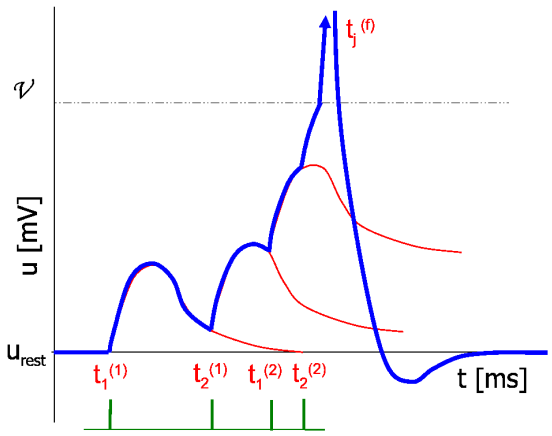


Fig. 2.3 The potential at the postsynaptic neuron

In this way, a rule $E/a^k \rightarrow a^p; d$ $(k > p)$ where a^p are p decaying spikes (a, e) can be seen with our notation as $E/a^k \rightarrow (a^p, S); d$ with $S = (s_1, \ldots, s_{e+2})$, $s_1 = k$, $s_2 = \cdots = s_{e+1} = p$ and $s_{e+2} = 0$.

2.3 Hebbian SN P System Units Hebbian SN P system units are SN P systems with a fixed topology endowed with input and decay. They have the following common features:

- The initial number of the spikes inside the neurons is always zero in all Hebbian SN P system units, so we do not refer to them in the description of the unit.
- All the presynaptic neurons are linked to only one postsynaptic neuron and these are all the synapses in the SN P system, so they are not provided in the description.
- The output neuron is the postsynaptic one.

Bearing in mind these features, we describe a Hebbian SN P system unit in the following way.

Definition 1 *A Hebbian SN P system unit of degree* (m, k, p) *is a construct*

$$H\Pi = (O, u_1, \ldots, u_m, v),$$

where:

- $O = \{a\}$ *is the alphabet (the object* a *is called spike);*

- $u_1, \ldots, u_m$ *are the presynaptic neurons. Each presynaptic neuron* u_i *has associated a set of rules* $R_i = \{R_{i1}, \ldots, R_{il_i}\}$ *where for each* $i \in \{1, \ldots, m\}$ *and* $j \in \{1, \ldots, l_i\}$, R_{ij} *is a decaying rule of the form:*

$$a^k \rightarrow (a^{n_{ij}}, S); d_{ij}$$

where $k \geq n_{ij} \geq 0$ *and* $d_{ij} \geq 0$. *We will call* n_{ij} *the* presynaptic potential *of the rule and* d_{ij} *is the* delay *of the rule. Note that all rules are triggered by* k *spikes. The decaying sequence* $S = (s_1, s_2, \ldots, s_r)$ *is a finite non increasing sequence of natural numbers called the decaying sequence where* $s_1 = k$ *and* $s_r = 0$.

- v *is the postsynaptic neuron which contains only one postsynaptic rule*

$$E_p^*/a^p \rightarrow a; 0$$

where E_p^* *is the set*[4] *of regular expressions* $\{n \in \mathbb{N} \,|\, n \geq p\}$. *We will call* p *the* threshold *of the postsynaptic potential of the Hebbian SN P system unit.*

By considering the decaying sequences we can distinguish among three types of Hebbian SN P system units:

- Hebbian SN P system units with *uniform decay*. In this case the decaying sequence S is the same for all the rules in the presynaptic neurons.
- Hebbian SN P system units with *locally uniform decay*. In this case the decaying sequence S is the same for all the rules in each presynaptic neuron.
- Hebbian SN P system units with *non-uniform decay*. In this case each rule has associated a decaying sequence.

Definition 2 *An* input *for a* Hebbian SN P system unit *of degree* m *is a vector* $\vec{x} = (x_1, \ldots, x_m)$ *of* m *non-negative integers* x_i.

A Hebbian SN P system unit with input *is a pair* $(H\Pi, \vec{x})$ *where* $H\Pi$ *is Hebbian SN P system unit and* $\vec{x}$ *is an input for it.*

The intuitive idea behind the input is encoding the information in time. Each component of the input represent the moment, according to the global clock, in which k spikes are provided to the corresponding presynaptic neuron.

2.4 How it works In this section we provide a description of the semantics of a Hebbian SN P system unit of degree (m, k, p). As we saw before, each x_i in the input $\vec{x} = (x_1, \ldots, x_m)$ represents the time in which k spikes are provided to the neuron

[4] This rule is an adaptation of the concept of a rule from an extended spiking neural P system with thresholds taken from [7].

u_i. At the moment x_i in which the spikes arrive to the neuron u_i one rule $(a^k \rightarrow (a^{n_{ij}}, S); d_{ij})$ is chosen in a non-deterministic way among the rules of the neuron.

Applying it means that k spikes are consumed and we look in $S = (s_1, \ldots, s_r)$ for the least l such that $n_{ij} \geq s_l$. Such s_l spikes are sent to the postsynaptic neurons according to the delay d_{ij} in the usual way, i.e., s_l spikes arrive to the postsynaptic neuron at the moment $x_i + d_{ij} + 1$. The decay of such spikes is determined by the decaying sequence. As we saw above, if the spikes are not consumed by the triggering of a rule in the postsynaptic neuron, they decay and at time $x_i + d_{ij} + 2$ we will consider that $s_l - s_{l+1}$ spikes have disappeared and we only have s_{l+1} spikes in the postsynaptic neuron. If the spikes are not consumed in the following steps by the triggering of a postsynaptic rule, at time $x_0 + d_{ij} + 1 + r - l$ the number of spikes will be decreased to $s_r = 0$ and the spikes are lost. Formally, if the chosen rule at the membrane i is $R_{ij} \equiv a^k \rightarrow (a^{n_{ij}}, S); d_{ij}$ with $S = (s_1, \ldots, s_r)$ and the rule is activated at time $t = x_i$, then the number of spikes sent by R_{ij} occurring in the postsynaptic neuron at time $t = x_i + d_{ij} + 1 + k$ is s_{k+k}, if $k \in \{0, \ldots, r - l\}$ and zero otherwise. The index l is the least index in $\{1, \ldots, r\}$ such that $n_{ij} \geq s_l$.

The potential on the postsynaptic neuron depends on the contributions of the chosen rules in the presynaptic neurons. Such rules send spikes that arrive to the postsynaptic neuron at different instants which depend on the input (the instant in which the presynaptic neuron is activated) and the delay of the chosen rule. The contribution of each rule to the postsynaptic neuron also changes along the time due to the decay.

Formally, the potential of the postsynaptic neuron in a given instant is a natural number calculated as a function R^* which depends on the time t, on the input $\vec{x}$ and on the rules chosen in each neuron $R^*(R_{1i_1}, \ldots, R_{mi_m}, \vec{x}, t) \in \mathbb{N}$. Such a natural number represents the number of the spikes at the moment t in the postsynaptic neuron and it is the result of adding the contributions of the rules $R_{1i_1}, \ldots, R_{mi_m}$.

The Hebbian SN P system unit produces an output if the rule of the postsynaptic neuron v, $E_p^*/a^p \rightarrow a$ is triggered, i.e., if at any moment t the amount of spikes in the postsynaptic neuron is greater than or equal to the threshold p, then the rule is activated and triggered. If there does not exist such t, then the Hebbian SN P system unit does not send any spike to the environment.

Bearing in mind the decay of the spikes in the postsynaptic neuron, if any spike has been sent out by the postsynaptic neuron after an appropriate number of steps, any spike will be sent to the environment. In fact, we have a lower bound for the number of steps in which the spike can be expelled, so we have a decision method to determine if the input $\vec{x}$ produces or not an output[5].

[5] A detailed description and some examples can be found in [12].

3 Learning

If we look at the Hebbian SN P system units as computational devices where the target is the transmission of information, we can consider that the device *successes* if a spike is sent to the environment and it *fails* if the spike is not sent. In this way, the lack of determinism in the choice of rules is a crucial point in the success of the devices because as we have seen above, if we provide several times the same input, the system can succeed or not.

In order to improve the design of these computational devices and in a narrow analogy with the Hebbian principle, we introduce the concept of *efficacy* in the Hebbian SN P system units. Such efficacy is quantified by endowing each rule with a *weight* that changes along the time, by depending on the contribution of the rule to the success of the device.

According to [8], in Hebbian learning, a synaptic weight is changed by a small amount if presynaptic spike arrival and postsynaptic firing *coincides*. This simultaneity constraint is implemented by considering a parameter s_{ij} which is the difference between the arrival of the contribution of the rule R_{ij} and the postsynaptic firing. Thus, the efficacy of the synapses such that its contributions arrive repeatedly shortly before a postsynaptic spike occurs is increased (see [14] and [3]). The weights of synapses such that their contributions arrive to the postsynaptic neuron *after* the postsynaptic spike is expelled are decreased (see [4] and [17]). This is basically the learning mechanism suggested in [18].

3.5 The Model In order to implement a learning algorithm in our Hebbian SN P system units, we need to extend it with a set of weights that measure the efficacy of the synapses. The meaning of the weights is quite natural and it fits into the the theory of artificial neural networks [13]. The amount of spikes that arrives to the postsynaptic neuron due to the rule R_{ij} depends on the *contribution* of each rule and also on the *efficacy* w_{ij} of the synapse. As usual in artificial neural networks, the final contribution will be the contribution sent by the rule multiplied by the efficacy w_{ij}.

We fix these concepts in the following definition.

Definition 3 *An extended Hebbian SN P system unit of degree m is a construct*

$$EH\Pi = (H\Pi, w_{11}, \ldots, w_{ml_m}),$$

where:

- *$H\Pi$ is a Hebbian SN P system unit of degree m and the rules of the presynaptic neuron u_i are $R_i = \{R_{i1}, \ldots, R_{il_i}\}$ with $i \in \{1, \ldots, m\}$.*
- *For each rule R_{ij} with $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, l_i\}$, w_{ij} is a real number which denotes the* initial weight *of the rule R_{ij}.*

Associating a weight to each rule means to consider an individual synapse for each rule instead of a synapse associated to the whole neuron. The idea of considering several synapses between two neurons is not new in computational neuron models. For example, in [20] the authors present a model for spatial and temporal pattern analysis via spiking neurons where several synapses are considered. The same idea had previously appeared in [8]. Considering several rules in a neuron and one synapse associated to each rule allows us to design an algorithm for changing the weight (the efficacy) of the synapse according to the result of the different inputs.

The concept of input of a extended Hebbian SN P system unit is similar to the previous one. The information is encoded in time and the input of each neuron denotes the moment in which the neuron is excited.

Definition 4 *An* extended Hebbian SN P system unit with input *is a pair* $(EH\Pi, \vec{x})$, *where* $H\Pi$ *is an Hebbian SN P system unit and* $\vec{x}$ *is an input for it.*

The semantics As we saw before, each x_i in the input $\vec{x} = (x_1, \ldots, x_m)$ represents the time in which the presynaptic neuron u_i is activated. The formalization of the activation of the neuron in this case differs from the Hebbian SN P system units. The idea behind the formalization is still the same: the postsynaptic neuron receives a little amount of electrical impulse according to the excitation time of the presynaptic neuron and the efficacy of the synapsis. The main difference is that we consider that there exist several synapses between one presynaptic neuron and the postsynaptic one (one synapse for each rule in the neuron) and the potential is transmitted along *all* these synapses according to their efficacy.

Extending the Hebbian SN P system units with efficacy in the synapses and considering that there are electrical flow along all of them can be seen as a generalisation of the Hebbian SN P system units. In Hebbian SN P system units only one rule R_{ij} is chosen in the presynaptic neuron u_i and the contribution emitted by R_{ij} arrives to the postsynaptic neuron according to the decaying sequence. Since the weight w_{ij} multiplies the contribution in order to compute the potential that arrives to the postsynaptic neuron, we can consider the Hebbian SN P system unit as an extended Hebbian SN P system unit with the weight of the chosen rule R_{ij} equals to one and the weight of the remaining rules equals to zero.

At the moment x_i in the presynaptic neuron u_i we will consider that *all* rules $(a^k \rightarrow (a^{n_{ij}}, S); d_{ij})$ are activated. The potential on the postsynaptic neuron depends on the contributions of the rules in the presynaptic neurons and the efficacy of the respective synapses. Let us consider that at time x_i the rule $(a^k \rightarrow (a^{n_{ij}}, S); d_{ij})$ is activated and the efficacy of its synapse is represented by the weight w_{ij}. When the rule $(a^k \rightarrow (a^{n_{ij}}, S); d_{ij})$ is triggered at the instant t_0 we look in $S = (s_1, \ldots, s_r)$ for the least l such that $p \times w_{ij} \geq s_l$. Then s_l spikes are sent to the postsynaptic neurons according with the delay d in the usual way.

At t_0+d+1, the s_l spikes arrive to the postsynaptic neurons. The decay of such spikes is determined by the decaying sequence. If the spikes are not consumed by the triggering of a rule in the postsynaptic neuron, they decay and at time $t_0 + d + 2$ we will consider that $s_l - s_{l+1}$ spikes have disappeared and we only have s_{l+1} spikes in the postsynaptic neuron. If the spikes are not consumed in the following steps by the triggering of a postsynaptic rule, at step $t_0 + d + 1 + r - l$ the number of spikes will be decreased to $s_r = 0$ and the spikes are lost. The extended Hebbian SN P system unit produces an output if the rule of the postsynaptic neuron v, $E_p^*/a^p \rightarrow a$ is triggered.

3.6 The Learning Problem Let us come back to the Hebbian SN P system units. In such units, provided an input $\vec{x}$, success can be reached or not (i.e., the postsynaptic rule is triggered or not) depending on the non-deterministically rules chosen. In this way, the choice of some rules is *better* than the choice of other ones, by considering that a rule is *better* than another if the choice of the former leads us to the success with a higher probability than the choice of the latter. Our target is to learn which are the best rules according to this criterion.

Formally, a *learning problem* is a 4-uple $(EH\Pi, X, L, \epsilon)$, where:

- $EH\Pi$ is an extended Hebbian SN P system unit.
- $X = \{\vec{x_1}, \ldots \vec{x_n}\}$ is a finite set of *inputs* of $EH\Pi$.
- $L : \mathbb{Z} \rightarrow \mathbb{Z}$ is a function from the set of integer numbers onto the set of integer numbers. It is called the *learning function*.
- ϵ is a positive constant called the *rate of learning*.

The *output* of a learning problem is an extended Hebbian SN P system unit.

Informal description of the algorithm Let us consider an extended Hebbian SN P system $EH\Pi$, a learning function $L : \mathbb{Z} \rightarrow \mathbb{Z}$ and a rate of learning ϵ. Let us consider an input $\vec{x}$ and we will denote by $t_{\vec{x}}$ the moment when the postsynaptic neuron reaches the potential for the trigger of the postsynaptic neuron. If such potential is not reached (and the postsynaptic neuron is not triggered) then $t_{\vec{x}} = \infty$.

On the other hand, for each rule $R_{ij} \equiv a^k \rightarrow (a^{n_{ij}}, S); d_{ij}$ of a presynaptic neuron we can compute the moment $t_{ij}^{\vec{x}}$ in which its contribution to the postsynaptic potential arrives to the postsynaptic neuron. It depends on the input $\vec{x}$ and the delay d_{ij} of the rule

$$t_{ij}^{\vec{x}} = \vec{x}_i + d_{ij} + 1$$

where $\vec{x}_i$ is the i-th component of $\vec{x}$.

We are interested in the influence of the rule R_{ij} on the triggering of the postsynaptic neuron. For that we need to know the difference between the arrival of the contribution $t_{ij}^{\vec{x}}$ and the moment $t_{\vec{x}}$ in which the postsynaptic neuron is activated.

For each rule R_{ij} and each input $\vec{x}$, such a difference is

$$s_{ij}^{\vec{x}} = t_{\vec{x}} - t_{ij}^{\vec{x}}$$

- If $s_{ij}^{\vec{x}} = 0$, then the postsynaptic neuron reaches the activation exactly in the instant in which the contribution of the rule R_{ij} arrives to it. This fact leads us to consider that the contribution of R_{ij} to the postsynaptic potential has had a big influence on the activation of the postsynaptic neuron.
- If $s_{ij}^{\vec{x}} > 0$ and it is *small*, then the postsynaptic neuron reaches the activation a bit later than the arrival of the contribution of the rule R_{ij} to it. This fact leads us to consider that the contribution of R_{ij} to the postsynaptic potential has influenced on the activation of the postsynaptic neuron due to the decay, but it is not so important as in the previous case.
- If $s_{ij}^{\vec{x}} < 0$ or $s_{ij}^{\vec{x}} > 0$ and it is not *small*, then the contribution of R_{ij} has no influence on the activation of the postsynaptic neuron.

The different interpretations of *small* or *big influence* are determined by the different *learning functions* $L : \mathbb{Z} \to \mathbb{Z}$. For each rule R_{ij} and each input $\vec{x}$, $L(s_{ij}^{\vec{x}}) \in \mathbb{Z}$ measures de degree of influence of the contribution of R_{ij} to the activation of the postsynaptic neuron produced by the input $\vec{x}$.

According to the principle of Hebbian learning, the efficacy of the synapses such that their contributions influence on the activation of the postsynaptic neuron must be increased. The weights of synapses such that their contributions have no influence on the activation of the postsynaptic neuron are decreased.

Formally, given an extended Hebbian SN P system $H\Pi$, a learning function $L : \mathbb{Z} \to \mathbb{Z}$, a rate of learning ϵ and an input $\vec{x}$ of $H\Pi$, the *learning algorithm* outputs a new extended Hebbian SN P system $H\Pi'$ which is equal to $H\Pi$, but the weights: each w_{ij} has been replaced by a new w'_{ij} according to

$$w'_{ij} = w_{ij} + \epsilon \cdot L(s_{ij}^{\vec{x}})$$

Depending on the sign of $L(s_{ij}^{\vec{x}})$, the rule R_{ij} will increase or decrease its efficacy. Note that $L(s_{ij}^{\vec{x}})$ is multiplied by the *rate of learning* ϵ. This rate of learning is usual in learning process in artificial neural networks. It is usually a small number which guarantees that the changes on the efficacy are not abrupt.

Finally, given a *learning problem* $(H\Pi, X, L, \epsilon)$, the learning algorithm takes $\vec{x} \in X$ and outputs $H\Pi'$. In the second step, the learning problem $(H\Pi', X - \{\vec{x}\}, L)$ is considered and we get a new $H\Pi'$. The process finishes when all the inputs has been consumed and the algorithm outputs the last extended SN P system unit.

The use of weights needs more discussion. The weights are defined as real numbers and membrane computing devices are discrete. If we want to deal with discrete computation in all the steps of the learning process we have to choose the parameters carefully. The following result gives a sufficient constraint for having an integer number of spikes at any moment.

Theorem 1 *Let a be the greatest¡ non-negative integer such that for all presynaptic potential n_{ij} there exists an integer x_{ij} such that $n_{ij} = x_{ij} \cdot 10^a$.*

Let b be the smallest non-negative integer such that for all initial weight w_{ij} and for the rate of learning ϵ there exist the integers k_{ij} and k such that $w_{ij} = k_{ij} \cdot 10^{-b}$ and $\epsilon = k \cdot 10^{-b}$.

If $a - b \geq 0$, then for all presynaptic potential n_{ij} and all the weights w obtained along the learning process, $n_{ij} \cdot w$ is an integer number.

Proof For the sake of simplicity, we denote by w^r the update weight w after r steps (and w^0 is the initial weight). Then, it suffices to consider the recursive generation of new weights $w^{n+1} = w^n + \epsilon \cdot L(s_n)$, where s_n is the corresponding value in the step n, and therefore

$$w^{n+1} = w^0 + \epsilon \cdot (L(s_0) + \cdots + L(s_n)).$$

If we develop $n_{ij} \cdot w^{n+1}$ according to the statement of the theorem, we have that there exist the integers x_{ij}, k_0 and k such that

$$\begin{aligned} n_{ij} \cdot w^{n+1} &= x_{ij} \cdot 10^a \cdot [k_0 \cdot 10^{-b} + (k \cdot 10^{-b}(L(s_0) + \cdots + L(s_n)))] \\ &= 10^{a-b} \cdot x_{ij} \cdot [k_0 + k(L(s_0) + \cdots + L(s_n))] \end{aligned}$$

Since $x_{ij} \cdot [k_0 + k(L(s_0) + \cdots + L(s_n))]$ is an integer number, if $a - b \geq 0$ then $n_{ij} \cdot w^{n+1}$ is an integer number. □

4 A Case Study

Let us consider the Hebbian SN P system

$$H\Pi = (O, u_1, u_2, v)$$

with uniform decay, where:

- $O = \{a\}$ is the alphabet;
- u_1, u_2 are the presynaptic neurons. The presynaptic neurons u_1, u_2 have associated the sets of rules R_i, where $R_1 = \{R_{11}, R_{12}, R_{13}\}$ and $R_2 = \{R_{21}, R_{22}\}$, respectively, with

$$R_{11} \equiv a^{3000} \to (a^{3000}, S); 0 \qquad R_{21} \equiv a^{3000} \to a^{1000}; 0$$
$$R_{12} \equiv a^{3000} \to (a^{2000}, S); 1 \qquad R_{22} \equiv a^{3000} \to a^{3000}; 3$$
$$R_{13} \equiv a^{3000} \to (a^{2000}, S); 7$$

- The *decaying sequence* is $S = (3000, 2800, 1000, 500, 0)$.
- v is the postsynaptic neuron which contains only one postsynaptic rule $E^*_{1200}/a^{1200} \to a; 0$.

Let $EH\Pi$ be the Hebbian SN P system unit $H\Pi$ extended with the initial weights $w_{11} = w_{12} = w_{13} = w_{21} = w_{22} = 0.5$.

Let us consider the learning problem $(EH\Pi, X, L, \epsilon)$ where

- $EH\Pi$ is the extended Hebbian SN P system unit described above,
- X is a set of 200 random inputs (x_i^1, x_i^2) with $1 \leq i \leq 200$ and $x_i^1, x_i^2 \in \{0, 1, \ldots, 5\}$
- L is the learning function $L : \mathbb{Z} \to \mathbb{Z}$

$$L(s) = \begin{cases} 3 & \text{if } s = 0 \\ 1 & \text{if } s = 1 \\ -1 & \text{otherwise} \end{cases}$$

- The rate of learning is $\epsilon = 0.001$

We have programmed an appropriate software for dealing with learning problems. After applying the learning algorithm, we obtain a new extended Hebbian SN P system unit similar to $EH\Pi$ but with the weights

$$w_{11} = 0.754, \quad w_{12} = 0.992, \quad w_{13} = 0.3, \quad w_{21} = 0.454, \quad w_{22} = 0.460$$

Figure 4.4 shows the evolution of the weights of the synapses.

The learning process shows clearly the differences among the rules.

- The *worst* rule is R_{13}. In a debugging process of the design of an SN P system network such rule should be removed. The value of the weight has decreased along all the learning process. This fact means that the rule has never contributed to the success of the unit and then it can be removed.
- On the contrary, the *best* rules are R_{11} and R_{12}. In most of the cases, (not all) these rules have been involved in the success of the unit.
- The other two rules R_{21} and R_{22} have eventually contributed to the success of the unit but not so clearly as R_{11} and R_{21}.

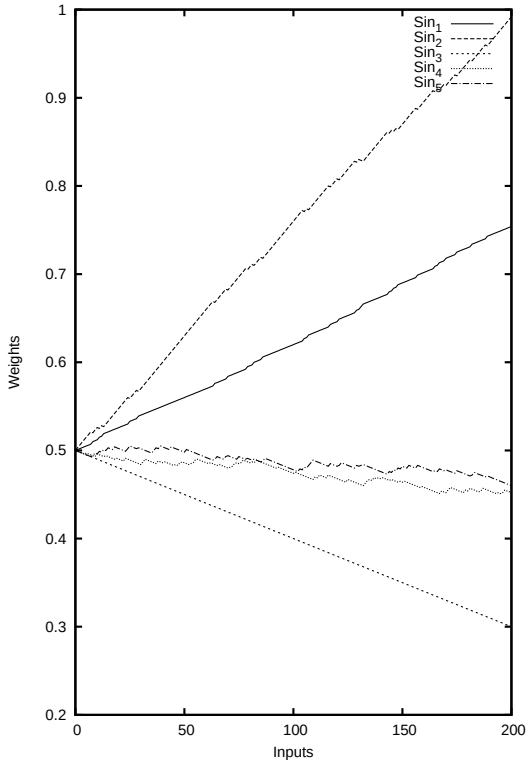


Fig. 4.4 The evolution of the weights

5 Conclusions and Future Work

The integration in a unique model of concepts from neuroscience, artificial neural networks and spiking neural P systems is not an easy task. Each of the three fields have its own concepts, languages and features. The work of integration consists on choosing ingredients from each field and trying to compose a computational device with the different parts. This means that some of the ingredients used in the devices presented in this paper are not usual in the SN P systems framework. Although the authors have tried to be as close to the SN P system spirit as possible some remarks should be considered.

In the paper, the input of the device is provided as a vector $(x_1, \ldots, x_m)$ of non-negative integers, where x_i represents the moment in which one rule (non-deterministically chosen) of the neuron u_i is activated. The information encoded in the vector $(x_1, \ldots, x_m)$ can be provided to the input neurons by m spike trains were all the elements are 0's and there is only one 1 in the position t_i. In this way, the input is encoded by m spike trains, which is closer to the standard inputs in SN P systems.

The idea of providing the input with a spike train of 0's and only one 1 in the position t_i carries out new problems. In the literature of SN P systems, in the instant t_i only one spike is supplied to the neuron u_i. In our device we want that a rule of type $a^r \rightarrow a^p; d$ is activated with $r > 1$. At this point we can consider several choices. The first one is to consider that at time t_i the spike train provides r spikes, but this choice lead us far from the SN P system theory. A second option is to consider that the spike trains have r consecutive 1's and each of them provide one spike. The remaining elements in the train are zeros. In this way the moment t_i will be the instant in which the r spikes have been provided to the neuron. A drawback for this proposal can be that r can be a big number and this increase the number of steps of the device. A third choice is to consider amplifier modules as in Figure 5.5. The leftmost neuron receives a spike train where all the elements are 0's but the $t_i - th$ which is 1. At the moment t_i only one spike is supplied to the neuron. At $t_i + 1$, one spike arrives to the r postsynaptic neurons, and each of them sends one spike to the rightmost neuron, so at $t_i + 2$ exactly r spikes arrive simultaneously to the last neuron.

These three solutions can be an alternative to the use of the vector $(t_1, \ldots, t_m)$ and deserve to be considered for further research in this topic.

Another main concept in this paper is the decay. It has strong biological intuition, but it is difficult to insert into the SN P systems theory. The main reason is that if we consider the spike as the information unit it does not make sense to talk about a half of a spike or a third of a spike. In that sense, the approach to decay from [7] is full of sense since one spike exists or it is lost, but its potential it is not decreasing in time.

The key point for the decay in this paper is taken from the definition of *extended* SN P systems. In such devices, a neuron can send a different amount of spikes depending on the chosen rule. So, the information is not only encoded in the *time* between two consecutive spikes, but on the *number* of spikes. This lead us to define the decay as a decrement in the number of spikes. In this way, we can consider that a pulse between two neurons is composed by a certain number of spikes which can be partially lost depending on the time.

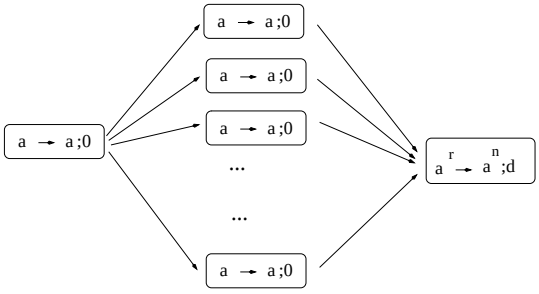


Fig. 5.5 Amplifier module

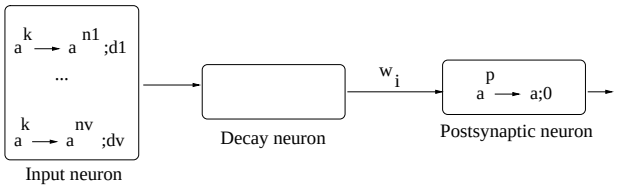


Fig. 5.6 Including a decay neuron

In this paper, such a decay has been implemented by extending the rules with a finite decreasing sequence which can be uniform, locally-uniform or non uniform for the set of rules. Other implementations are also possible. Probably, the decay can also be implemented with an extra neuron as in Figure 5.6 which sends to the final neuron a decaying sequence of spikes.

The use of weights also deserves to be discussed. In Theorem 1 we provide sufficient conditions for handling at every moment an integer number of spikes. Nonetheless, further questions should be considered. For example, the use of negative weights or weights greater than one. Should we consider negative weights and/or a *negative* contribution to the postsynaptic potential? The use weights greater than one leads us to consider that the contribution of one rule to the postsynaptic potential is *greater than* its own presynaptic potential. Can the efficiency of the synapses amplify the potential beyond the number of emitted spikes?

More technical questions are related to the rate of learning and to the algorithm of learning. Both concepts have been directly borrowed from artificial neural networks and need deeper study in order to adapt them to the specific features of SN P systems.

As a final remark, we consider that this paper opens a promising line research bridging SN P systems and artificial neural networks without forgetting the biological inspiration and also opens a door to applications of SN P systems.

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

Bibliography

[1] E.D. Adrian. The impulses produced by sensory nerve endings: Part I. *The Journal of Physiology* 61, (1926), 49–72.
[2] E.D. Adrian. *The basis of Sensation.* W.W. Norton. New York, (1926).
[3] T.V.P. Bliss and G.L. Collingridge. *Nature*, 361, (1993), 31–99.
[4] D. Debanne, B.H. Gähwiler and S.M. Thompson. Asynchronous Pre- and Postsynaptic Activity Induces Associative Long-Term Depression in Area CA1 of the Rat

Hippocampus in vitro. *Proceedings of the National Academy of Sciences*, 91(3), (1994), 1148–1152.

[5] R. Eckhorn, R. Bauer, W. Jordan, M. Brosch, W. Kruse, M. Munk and H.J. Reitboeck. Coherent oscillations: A mechanism of feature linking in the visual cortex? *Biological Cybernetics* 60, (1998), 121–130.

[6] A.K. Engel, P. König, A.K. Kreiter, and W. Singer. Interhemispheric sycrhonization of oscillatory neural responses in cat visual cortex. *Science*, 252, (1991), 1177–1179.

[7] R. Freund, M. Ionescu and M. Oswald. Extended spiking neural P systems with decaying spikes and/or total spiking. *Proceedings of the International Workshop Automata for Cellular and Molecular Computing*, MTA SZTAKI, Budapest, (2007), 64–75.

[8] W. Gerstner, R. Kempter, L. van Hemmen and H. Wagner. A neuronal learning rule for sub-millisecond temporal coding. *Nature*, 383, (1996), 76–78.

[9] W. Gerstner and W.Kistler. *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge University Press, (2002).

[10] C.M. Gray, P. König, A.K. Engel and W. Singer. Oscillatory responses in cat visual cortex exhibit inter-columnar synchronization which reflects global stimulus properties. *Nature*, 338, (1989), 334–337.

[11] C.M. Gray and W. Singer. Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex. *Proceedings of the National Academy of Sciences*, 86, (1989), 1698–1702.

[12] M.A. Gutiérrez- Naranjo and M.J. Pérez-Jiménez. A First Model for Hebbian Learning with Spiking Neural P Systems. In D. Díaz-Pernil *et al.* (eds.), *Sixth Brainstorming Week on Membrane Computing*. Fénix Editora, (2008), 211-233.

[13] S. Haykin. *Neural Networks. A Comprehensive Foundation*. Macmillan College Publishin Company, Inc. (1994).

[14] D.O. Hebb. *The Organization of Behavior*. Wiley, New York, (1949).

[15] D.H. Hubel and T.N. Wiesel. Receptive Fields of Single Neurons in the Cat's Striate Cortex. *The Journal of Phisiology*, 148, (1959), 574–591.

[16] M. Ionescu, Gh. Păun and T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71 (2-3), (2006), 279–308.

[17] H. Markram and B. Sakmann. Action potentials propagating back into dendrites triggers changes in efficiency of single-axon synapses between layer V pyramidal neurons. *Soc. Neurosci. Abstr.*, 21, (1995), 2007.

[18] H. Markram and M. Tsodyks. Redistribution of synaptic efficacy between neocortical pyramidal neurons. *Nature*, 382, (1996), 807–810.

[19] V.B. Mountcastle. Modality anf topographic properties of single neurons of cat's somatosensory cortex. *Journal of Neurophysiology*, 20, (1957), 408–434.

[20] T. Natschläger and B. Ruf. Spatial and temporal pattern analysis via spiking neurons. *Network: Computation in Neural Systems*, 9(3), (1998), 319–338.

[21] Gh. Păun: *Membrane Computing. An Introduction*. Springer–Verlag, Berlin, 2002.

[22] Gh. Păun: Twenty six research topics about spiking neural P systems. In M.A. Gutiérrez-Naranjo, Gh. Păun, A. Romero-Jiménez and A. Riscos-Núñez, editors.

Fifth Brainstorming Week on Membrane Computing, Fénix Editora, Sevilla, (2007), 263–280.

[23] S. Ramón y Cajal. *Histologie du Systeme Nerveux de l'Homme et des Vertebre*. A. Maloine, Paris, (1909).

[24] W. Singer. The role of synchrony in neocortical processing and synaptic plasticity. In E. Domany, J.L. van Hemmel and K. Schulten, editors, *Models of Neural Networks II*, chapter 4, Springer Verlag, Berlin, (1994).

[25] S. Thorpe, S, Fize and C. Marlot. Speed of processing in the human visual system. *Nature*, 381, (1996), 520-522.

[26] P systems web page `http://ppage.psystems.eu/`

Event-driven Metamorphoses of P Systems

Thomas Hinze, Raffael Faßler, Thorsten Lenser, Naoki Matsumaru, Peter Dittrich

Friedrich-Schiller-Universität Jena, Bio Systems Analysis Group,
Ernst-Abbe-Platz 1–4, D-07743 Jena, Germany
{hinze,raf,thlenser,naoki,dittrich}@minet.uni-jena.de

Complex reaction systems in molecular biology are often composed of partially independent subsystems associated with the activity of external or internal triggers. Occurring as random events or dedicated physical signals, triggers effect transitions from one subsystem to another which might result in substantial changes of detectable behaviour. From a modelling point of view, those subsystems typically differ in their reaction rules or principle of operation. We propose a formulation of trigger-based switching between models from a class of P systems with progression in time employing discretised mass-action kinetics. Two examples inspired by biological phenomena illustrate the consecutive interplay of P systems towards structural plasticity in reaction rules: evolutionary construction of reaction networks and artificial chemistries with self-reproducible subunits.

1 Introduction

Structural dynamics in biological reaction networks, also known as *plasticity* [4, 21], is a common property of complex processes in living systems and their evolution. Within the last years, its impetus for adaptation, variability, emergence, and advancement in biology became more and more obvious. Facets of life provide a plethora of examples for structural dynamics: Organisms undergo *metamorphosis* by the physical development of their form and metabolism. At a more specific level, synaptic plasticity within central nervous systems of animals [6] as well as photosynthesis of plants [3] are characterised by substantial structural changes of the underlying reaction networks over time, depending on external or even internal signals. In case of photosynthesis, light-dependent reactions differ from processes of the Calvin cycle. In nervous systems, presence of neurotransmitters for longer periods effects duplication or discreation of synapses. All these and many further biological phenomena can be divided into several stages of function. Typically, each stage corresponds to a subsystem of fixed components. In terms of modelling aspects, such a subsystem is defined by a dedicated set of species and unchanging reactions. Only the species concentrations vary in time or space according to identical rules.

Along with the development of systems biology, the integration of separately considered subsystems into more general frameworks comes increasingly into the focus of research to understand complex biological phenomena as a whole [1]. From this perspective, the question arises how to assemble multiple process models, each of which captures selected specific aspects of the overall system behaviour.

Motivated by this question, we contribute to the specification of a framework able to incorporate correlated temporally "local" models whose dynamical behaviour passes over from one to the other. In this context, time- and value-discrete approaches promise a high degree of flexibility in separate handling of atomic objects rather than analytical methods since singularities caused by transition between models can affect continuous gradients and amplify numerical deviations. We introduce a deterministic class Π_{PMA} of P systems with strict *prioritisation* of reaction rules and a principle of operation based on discretised *mass-action* kinetics. Systems within this class enable an iterative progression in time. Representing temporally local models of chemical reaction systems, they are designed to interface to each other. An overlying state transition system manages the structural dynamics of P systems Π_{PMA} according to signals mathematically encoded by constraints (boolean expressions). Two case studies gain insight into the descriptional capabilities of this framework.

Related work addresses two aspects: discretisation of chemical kinetics and structural network dynamics. On the one hand, metabolic or cell signalling P systems like [14] describe the dynamical behaviour of a fixed reaction network based on concentration gradients, numerically studied in [8]. Results of [10] present a discretisation of Hill kinetics mainly employed for gene regulatory networks. Artificial chemistries were explored in [7] along with issues of computability [13] and prioritisation of reaction rules [20]. On the other hand, spatial structural dynamics in P systems was primarily considered as active membranes [16, 17]. Dynamical reaction rules in probabilistic P systems were investigated in [18].

The paper is organised as follows: First we present a method for discretisation of mass-action kinetics leading to P systems Π_{PMA} whose properties are discussed briefly. Section 3 introduces a transition framework for P systems of this class together with a description of the transition process. For demonstration, a chemical register machine with self-reproducible components for bit storage is formulated and simulated in Section 4. Two specialities are utilisation of a chemical clock based on an oscillating reaction network and a chemical encoding of binary numbers for register contents ensuring a high reliability in function from an engineering point of view. In Section 5, we discuss the transition framework for monitoring the evolutionary construction of reaction networks.

2 Deterministic P Systems for Chemistries Based on Mass-Action Kinetics

Multiset Prerequisites. Let A be an arbitrary set and $\mathbb{N}$ the set of natural numbers including zero. A multiset over A is a mapping $F : A \longrightarrow \mathbb{N} \cup \{\infty\}$. $F(a)$, also denoted as $[a]_F$, specifies the multiplicity of $a \in A$ in F. Multisets can be written as an elementwise enumeration of the form $\{(a_1, F(a_1)), (a_2, F(a_2)), \ldots\}$ since $\forall (a, b_1), (a, b_2) \in F : b_1 = b_2$. The support of F, $\mathrm{supp}(F) \subseteq A$, is defined by $\mathrm{supp}(F) = \{a \in A \mid F(a) > 0\}$. A multiset F over A is said to be empty iff $\forall a \in A : F(a) = 0$. The cardinality $|F|$ of F over A is $|F| = \sum_{a \in A} F(a)$. Let F_1 and F_2 be multisets over A. F_1 is a subset of F_2, denoted as $F_1 \subseteq F_2$, iff $\forall a \in A : (F_1(a) \leq F_2(a))$. Multisets F_1 and F_2 are equal iff $F_1 \subseteq F_2 \wedge F_2 \subseteq F_1$. The intersection $F_1 \cap F_2 = \{(a, F(a)) \mid a \in A \wedge F(a) = \min(F_1(a), F_2(a))\}$, the multiset sum $F_1 \uplus F_2 = \{(a, F(a)) \mid a \in A \wedge F(a) = F_1(a) + F_2(a)\}$, and the multiset difference $F_1 \ominus F_2 = \{(a, F(a)) \mid a \in A \wedge F(a) = \max(F_1(a) - F_2(a), 0)\}$ form multiset operations. The term $\langle A \rangle = \{F : A \longrightarrow \mathbb{N} \cup \{\infty\}\}$ describes the set of all multisets over A.

Mass-Action Kinetics for Chemical Reactions. The dynamical behaviour of chemical reaction networks is described by the species concentrations over the time course. According to biologically predefined motifs, a variety of models exists to formulate the reaction kinetics. Since most of them imply specific assumptions, we restrict ourselves to general mass-action kinetics [5]. Here, a continuous approach to express the dynamical behaviour considers production and consumption rates v_p and v_c of each species S in order to change its concentration by $\frac{\mathrm{d}\,[S]}{\mathrm{d}\,t} = v_p([S]) - v_c([S])$. These rates result from the reactant concentrations, their stoichiometric factors $a_{i,j} \in \mathbb{N}$ (reactants), $b_{i,j} \in \mathbb{N}$ (products) and kinetic constants $\hat{k}_j \in \mathbb{R}_+$ assigned to each reaction quantifying its speed. For a reaction system with a total number of n species and h reactions

$$
\begin{aligned}
a_{1,1}\mathrm{S}_1 + a_{2,1}\mathrm{S}_2 + \ldots + a_{n,1}\mathrm{S}_n &\xrightarrow{\hat{k}_1} b_{1,1}\mathrm{S}_1 + b_{2,1}\mathrm{S}_2 + \ldots + b_{n,1}\mathrm{S}_n \\
a_{1,2}\mathrm{S}_1 + a_{2,2}\mathrm{S}_2 + \ldots + a_{n,2}\mathrm{S}_n &\xrightarrow{\hat{k}_2} b_{1,2}\mathrm{S}_1 + b_{2,2}\mathrm{S}_2 + \ldots + b_{n,2}\mathrm{S}_n \\
&\vdots \\
a_{1,h}\mathrm{S}_1 + a_{2,h}\mathrm{S}_2 + \ldots + a_{n,h}\mathrm{S}_n &\xrightarrow{\hat{k}_h} b_{1,h}\mathrm{S}_1 + b_{2,h}\mathrm{S}_2 + \ldots + b_{n,h}\mathrm{S}_n,
\end{aligned}
$$

the corresponding ordinary differential equations (ODEs) read [7]:

$$\frac{\mathrm{d}\,[S_i]}{\mathrm{d}\,t} = \sum_{\nu=1}^{h} \left(\hat{k}_\nu \cdot (b_{i,\nu} - a_{i,\nu}) \cdot \prod_{l=1}^{n} [S_l]^{a_{l,\nu}} \right) \quad \text{with} \quad i = 1, \ldots, n.$$

In order to obtain a concrete trajectory, all initial concentrations $[S_i](0) \in \mathbb{R}_+$, $i = 1, \ldots, n$ are allowed to be set according to the needs of the reaction system.

Discretisation: Corresponding P Systems Π_{PMA}. The general form of a P system Π_{PMA} emulating the dynamical behaviour of chemical reaction systems with strict *prioritisation* of reaction rules based on discretised *mass-action* kinetics is a construct

$$\Pi_{\mathrm{PMA}} = (V, \Sigma, [_1]_1, L_0, R, K)$$

where V denotes the system alphabet containing symbol objects (molecular species) and $\Sigma \subseteq V$ represents the terminal alphabet. Π_{PMA} does not incorporate inner membranes, so the only membrane is the skin membrane $[_1]_1$. The single membrane property results from the assumption of spatial globality in well-stirred reaction vessels. Within a single vessel, the finite multiset $L_0 \subset V \times (\mathbb{N} \cup \{\infty\})$ holds the initial configuration of the system.

We formulate reaction rules together with their kinetic constants by the system components R and K. The finite set $R = \{r_1, \ldots, r_h\}$ subsumes the reaction rules while each reaction rule $r_i \in \langle E_i \rangle \times \langle P_i \rangle$ is composed of a finite multiset of reactants (educts) $E_i \subset V \times \mathbb{N}$ and products $P_i \subset V \times \mathbb{N}$. Multiplicities of elements correspond with according stoichiometric factors. Furthermore, a kinetic constant $k_i \in \mathbb{R}_+$ is attached to each reaction r_i forming $K = \{k_1, \ldots, k_h\}$.

Since we strive for a deterministic P system, a strict prioritisation among reaction rules is introduced in order to avoid conflicts that can appear if the amount of molecules in the vessel is too low to satisfy all matching reactions. In this case, running all matching reactions in parallel can lead to the unwanted effect that more reactant molecules are taken from the vessel than available violating conservation of mass. Prioritisation provides one possible strategy to select applicable reaction rules in contrast to random decisions (introduction of stochasticity) or separate consideration of the combinatorial variety (nondeterministic tracing). For large amounts of molecules in the vessel, the strategy of conflict handling has no influence to the dynamical system behaviour and can be neglected. We define the priority of a reaction rule by its index: $r_1 > r_2 > \ldots > r_h$.

For better readability, we subsequently write a reaction rule $r_i = \Big(\{(e_1, a_1), \ldots, (e_\mu, a_\mu)\}, \{(q_1, b_1), \ldots, (q_v, b_v)\} \Big)$ with $\mathrm{supp}(E_i) = \{e_1, \ldots, e_\mu\}$ and $\mathrm{supp}(P_i) = \{q_1, \ldots, q_v\}$ as well as kinetic constant k_i by using the chemical denotation $r_i : a_1\ e_1 + \ldots + a_\mu\ e_\mu \xrightarrow{k_i} b_1\ q_1 + \ldots + b_v\ q_v$.

Finally, the dynamical behaviour of P systems of the form Π_{PMA} is specified by an iteration scheme updating the system configuration L_t at discrete points in time starting from the initial configuration L_0 whereas a second index $i = 1, \ldots, h$ reflects intermediate phases addressing the progress in employing reactions:

$$
\begin{aligned}
L_{t,0} &= L_t \\
L_{t,i} &= L_{t,i-1} \ominus \Bigg\{ \Bigg(a, \Bigg\lfloor k_i \cdot |E_i \cap \{(a,\infty)\}| \cdot \\
&\qquad \prod_{b \in \mathrm{supp}(E_i)} |L_{t,i-1} \cap \{(b,\infty)\}|^{|E_i \cap \{(b,\infty)\}|} \Bigg\rfloor \Bigg) \;\Bigg|\; \forall a \in \mathrm{supp}(E_i) \\
&\qquad\qquad \wedge \left(|L_{t,i-1} \cap \{(c,\infty)\}| \geq |E_i \cap \{(c,\infty)\}| \;\forall c \in \mathrm{supp}(E_i) \right) \Bigg\} \\
&\uplus \Bigg\{ \Bigg(a, \Bigg\lfloor k_i \cdot |P_i \cap \{(a,\infty)\}| \cdot \\
&\qquad \prod_{b \in \mathrm{supp}(E_i)} |L_{t,i-1} \cap \{(b,\infty)\}|^{|E_i \cap \{(b,\infty)\}|} \Bigg\rfloor \Bigg) \;\Bigg|\; \forall a \in \mathrm{supp}(P_i) \\
&\qquad\qquad \wedge \left(|L_{t,i-1} \cap \{(c,\infty)\}| \geq |E_i \cap \{(c,\infty)\}| \;\forall c \in \mathrm{supp}(E_i) \right) \Bigg\} \\
L_{t+1} &= L_{t,h}.
\end{aligned}
$$

The iteration scheme modifies the system configuration by successive application of reaction rules according to their priority in two stages. The first stage identifies the reactants of a reaction. For this purpose, the required amount of each reactant molecule $a \in \mathrm{supp}(E_i)$ is determined by the kinetic constant k_i, the stoichiometric factor of a (obtained by $|E_i \cap \{(a,\infty)\}|$), and the product of all discretised reactant concentrations. Therefore, the term $|L_{t,i-1} \cap \{(b,\infty)\}|$ describes the number of molecules b currently available in the vessel. Since a reaction is allowed to become employed if and only if it can be satisfied, the constraint $|L_{t,i-1} \cap \{(c,\infty)\}| \geq |E_i \cap \{(c,\infty)\}| \;\forall c \in \mathrm{supp}(E_i)$ checks this property. Along with removal of reactant molecules (multiset difference $\ominus$), corresponding product molecules are added ($\uplus$) to obtain the intermediate configuration $L_{t,i}$ after taking reactions $r_1, \ldots, r_i$ into consideration.

In order to adapt reaction rates for discretisation, primary kinetic constants $\hat{k}_i$ defined in the continuous ODE approach have to be converted into counterparts for the discrete iteration scheme by using the transformation

$$k_i = \frac{\hat{k}_i}{\mathrm{V}^{|E_i|}} \cdot \Delta t.$$

Here, $\mathrm{V} \in \mathbb{R}_+ \setminus \{0\}$ expresses the volume of the reaction vessel while the exponent $|E_i|$ declares the sum of all stoichiometric factors of reactants occurring in reaction r_i. Constant $\Delta t \in \mathbb{R}_+ \setminus \{0\}$ specifies the discretisation interval.

In each time step, the P system Π_{PMA} generates the multiset output

$$O_t = L_t \cap \{(w, \infty) \mid w \in \Sigma\}$$

which can either be interpreted as natural numbers $n_t = |O_t|$ over time or contribute to the formal language $L(\Pi_{\mathrm{PMA}}) = \mathrm{supp}\,(\biguplus_{t=0}^{\infty} O_t) \subseteq \Sigma$.

System Classification and Properties. Π_{PMA} belongs to deterministic P systems with symbol objects, strict prioritisation of reaction rules, and progression in time according to mass-action kinetics that is time- and value-discretely approximated by a stepwise adaptation. Its principle of operation follows the idea of formulating one-vessel reaction systems together with their dynamical behaviour.

Obviously, P systems Π_{PMA} can emulate finite automata M. To this end, each transition $(q, a) \mapsto q'$ is transformed into a reaction $q + a \xrightarrow{1} q' + a$. Taking all final states as terminal alphabet, $L(\Pi_{\mathrm{PMA}}) = \emptyset$ iff $L(M) = \emptyset$ holds.

From the perspective of computational completeness, P systems Π_{PMA} as defined before cannot reach Turing universality: Although system configurations might represent any natural number, we need to define an explicit control mechanism able to address dedicated items (configuration components) for arbitrary incrementation, decrementation, and comparison to zero. Due to definition of mass-action kinetics, the number of molecules processed within one application of a reaction rule depends on the total amount of these molecules in the whole system. It seems that reaction rules should be *variable* during system evolution in order to enable enough flexibility. Allowing dynamical changes of kinetic constants or stoichiometric factors along with addition/deletion of reactions provides this flexibility, see Section 4.

3 Transitions between P Systems Π_{PMA}

In this section, we describe a framework enabling transitions between deterministic P systems Π_{PMA} on the fly. Initiated by an external trigger at a defined point in time, a transition manages three switching activities: Firstly, the running system stops its evolution. Secondly, the (only) resulting configuration of that system is mapped into the initial configuration of the subsequent system. This includes integration of possibly new species together with their initial number of copies put into the vessel as well as removal of vanished species iff specified. Reactions in R and kinetic parameters in K are replaced. Thirdly, the subsequent system is set into operation.

For formulation of the transition framework, we utilise *state transition systems* [19] denoted as construct $\mathcal{A} = (Q, T, I, \Delta, F)$ with a set Q of states (not necessarily finite but enumerable), an alphabet T of input symbols, a set $I \subseteq Q$ of initial states, the transition relation $\Delta \subseteq Q \times T \times Q$, and a set $F \subseteq Q$ of final states. In general, state transition systems are known to be nondeterministic allowing multiple transitions. For our objective, we arrange the components as follows:

$$
\begin{aligned}
Q &= \{\Pi^{(j)}_{\mathrm{PMA}} \mid (j \in A) \wedge (A \subseteq \mathbb{N})\} \\
T &\subseteq \{(t = \tau) \mid (\tau \in B) \wedge (B \subseteq \mathbb{N})\} \cup \\
&\quad \{([a]\, \mathit{CMP}\, \kappa) \mid (\mathit{CMP} \in \{<, \leq, =, \neq, \geq, >\}) \wedge (\kappa \in \mathbb{N}) \wedge (a \in V^{(j)}) \wedge \\
&\qquad (\Pi^{(j)}_{\mathrm{PMA}} = (V^{(j)}, \Sigma^{(j)}, [\,]_1, L^{(j)}_0, R^{(j)}, K^{(j)}) \in Q) \wedge (j \in A)\}.
\end{aligned}
$$

While each state in Q is represented by a dedicated P system Π_{PMA}, the input alphabet T contains a number of constraints (triggering events) with regard to progress in operation time $(t = \tau)$ or achievement of designated molecular amounts $([a]\, \mathit{CMP}\, \kappa)$. We assume that these constraints are related to the P system in Q currently in operation.

Each transition $\Pi^{(j)}_{\mathrm{PMA}} \overset{c}{\mapsto} \Pi^{(m)}_{\mathrm{PMA}} \in \Delta$ from $\Pi^{(j)}_{\mathrm{PMA}} = (V^{(j)}, \Sigma^{(j)}, [\,]_1, L^{(j)}_0, R^{(j)}, K^{(j)})$ to $\Pi^{(m)}_{\mathrm{PMA}} = (V^{(m)}, \Sigma^{(m)}, [\,]_1, L^{(m)}_0, R^{(m)}, K^{(m)})$ triggered by $c \in T$ allows addition of new species to system alphabets $V^{(j)}$ and $\Sigma^{(j)}$. Here, added species form sets $\mathit{AdditionalSpecies}V_{(j,m)}$ and $\mathit{AdditionalSpecies}\Sigma_{(j,m)}$ with $\mathit{AdditionalSpecies}V_{(j,m)} \cap V^{(j)} = \emptyset$ and $\mathit{AdditionalSpecies}\Sigma_{(j,m)} \cap \Sigma^{(j)} = \emptyset$. Furthermore, a species a is allowed to vanish if and only if $[a] = 0$. Corresponding sets $\mathit{VanishedSpecies}V_{(j,m)} \subset V^{(j)}$ and $\mathit{VanishedSpecies}\Sigma_{(j,m)} \subset \Sigma^{(j)}$ contain vanishing species. Within a transition $\Pi^{(j)}_{\mathrm{PMA}} \overset{c}{\mapsto} \Pi^{(m)}_{\mathrm{PMA}}$, new reactions might appear restricted to reactants and products available in $V^{(m)}$. New reactions $r_i \in \langle V^{(m)} \times \mathbb{N} \rangle \times \langle V^{(m)} \times \mathbb{N} \rangle$ with unique priority index i become accumulated by the multiset $\mathit{AdditionalReactions}_{(j,m)}$ together with assigned kinetic constants $k_i \in \mathbb{R}_+$ subsumed in the set $\mathit{ParsAdditionalReactions}_{(j,m)}$. Accordingly, we consider vanishing reactions present in multiset $\mathit{VanishedReactions}_{(j,m)}$ and their kinetic parameters stored in $\mathit{ParsVanishedReactions}_{(j,m)}$). The scheme

$$
\begin{aligned}
V^{(m)} &= V^{(j)} \cup \mathit{AdditionalSpecies}V_{(j,m)} \setminus \mathit{VanishedSpecies}V_{(j,m)} \\
\Sigma^{(m)} &= \Sigma^{(j)} \cup \mathit{AdditionalSpecies}\Sigma_{(j,m)} \setminus \mathit{VanishedSpecies}\Sigma_{(j,m)} \\
L^{(m)}_0 &= L^{(j)}_t \uplus \{(a, 0) \mid a \in \mathit{AdditionalSpecies}V_{(j,m)}\} \\
R^{(m)} &= R^{(j)} \uplus \mathit{AdditionalReactions}_{(j,m)} \ominus \mathit{VanishedReactions}_{(j,m)} \\
K^{(m)} &= K^{(j)} \cup \mathit{ParsAdditionalReactions}_{(j,m)} \setminus \mathit{ParsVanishedReactions}_{(j,m)}
\end{aligned}
$$

decomposes the P system transition into all single components. After performing the transition, the obtained system $\Pi^{(m)}_{\mathrm{PMA}}$ includes reactions $R^{(m)} = \{r_i \mid (i \in A) \wedge (A \subset$

$\mathbb{N})\}$ and corresponding kinetic parameters $K^{(m)} = \{k_i \mid (i \in A) \wedge (A \subset \mathbb{N})\}$ where A is an arbitrary finite subset of natural numbers. In order to preserve the strict prioritisation among reaction rules, pairwise distinctive indexes i are required in each set.

4 Chemical Register Machines with Self-Reproducible Components

In the first example, we apply transitions between P systems to formulate a chemical register machine on binary numbers with self-reproducible components for bit storage units. Each time new storing capacity within a register is needed, a specific reaction subsystem for that purpose is added. A strict modularisation of the reaction network forming bit storage units (chemical implementation of master-slave flip-flops) facilitates the system design towards achieving computational completeness. A chemical representation of binary numbers noticeably increases the reliability of operation from an engineering point of view.

Register Machines on Binary Numbers. A register machine on binary numbers is a tuple $M = (R, L, P, \#_0)$ consisting of the finite set of registers $R = \{\mathrm{R}_1, \ldots, \mathrm{R}_m\}$ each with binary representation of a natural number $\mathrm{R}_h \in \{0,1\}^*$, a finite set of jump labels (addresses) $L = \{\#_0, \ldots, \#_n\}$, a finite set P of instructions, and the jump label of the initial instruction $\#_0 \in L$. Available instructions are: $\#_i :$ INC R_h $\#_j$ (increment register R_h and jump to $\#_j$), $\#_i :$ DEC R_h $\#_j$ (nonnegatively decrement register R_h and jump to $\#_j$), $\#_i :$ IFZ R_h $\#_j$ $\#_p$ (if $\mathrm{R}_h = 0$ then jump to $\#_j$ else jump to $\#_p$), and $\#_i$ HALT (terminate program and output register contents). We assume a pre-initialisation of input and auxiliary registers at start with input data or zero. Furthermore, a deterministic principle of operation, expressed by unique usage of instruction labels: $\forall p, q \in P \mid (p = \#_i : v) \wedge (q = \#_j : w) \wedge ((i \neq j) \vee (v = w))$, is supposed.

Chemical Encoding of Binary Values. Each boolean variable $x \in \{0,1\}$ is represented by two correlated species X^T and X^F with complement concentrations $[X^T] \in \mathbb{R}_+$ and $[X^F] \in \mathbb{R}_+$ such that $[X^T]+[X^F] = c$ holds with $c = const$. The boolean value of the variable x is determined whenever one of the following conditions is fulfilled: The inequality $[X^T][\,[X^F]$ indicates "false" ($x = 0$) and $[X^F][\,[X^T]$ "true" ($x = 1$). In case of none of these strong inequalities holds (e.g. $[X^T] = 0.6c$ and $[X^F] = 0.4c$), the system would consider the variable x to be in both states.

A Chemical Clock by Extending an Oscillating Reaction Network. A chemical implementation of a clock is necessary in order to synchronise the register machine instruction processing. Positive edges of clock signals can trigger micro-operations like register increment or jump to the next machine instruction. In our chemical machine model, an

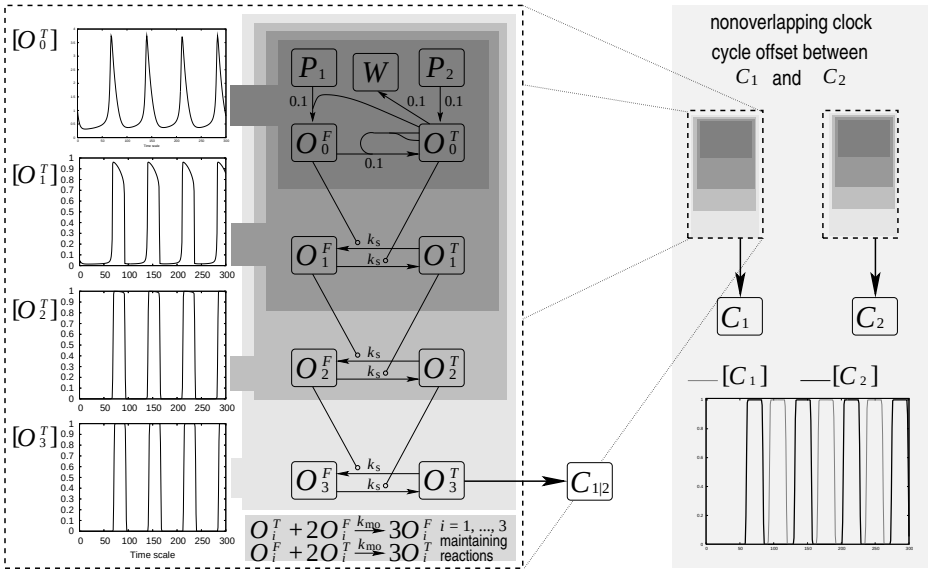


Fig. 4.1 Generation of chemical clock signals $[C_1]$, $[C_2]$ (right) by cascadisation of toggle switches (left)

extended oscillating reaction network provides all clock signals. As preferred network template for permanent oscillation, we adopt the well-studied Belousov-Zhabotinsky reaction [2, 22] depicted in the upper-left part of Figure 4.1 whose dynamical behaviour results in periodic peak-shaped signals. By using a cascade of downstream switching and maintaining reactions, we extend that primary oscillator. In this way, a normalisation with respect to signal shape and concentration course can be reached. Our idea employs both converse output signals O_i^T and O_i^F of the previous cascade stage as triggers for a subsequent chemical toggle switch. Thus, high and low concentration levels are more and more precisely separated over the time course, and the switching delay in between becomes shortened, see lower-left parts of Figure 4.1. After three cascade stages, the quality of the chemical clock signal turns out to be suitable for our purposes.

For technical reasons (two-phase register machine instruction processing), two offset clocks with designated output species C_1 and C_2 are employed. Owning the same network structure, they only differ in the time point when coming into operation caused by individual initialisations (species producing clock signals C_1: $[O^F_{0,C_1}](0) = 2$, $[O^T_{0,C_1}](0) = 1$; corresponding species for clock signals C_2: $[O^F_{0,C_2}](0) = 0, [O^T_{0,C_2}](0) = 0$; species with identical initial concentrations: $[P_{1,C}](0) = 3, [P_{2,C}](0) = 1, [W_C](0) = 0, [O^F_{i,C}](0) = 1, [O^T_{i,C}](0) = 0), i \in \{1, 2, 3\}, C \in \{C_1, C_2\}$). C_1 and C_2 provide nonoverlapping clock signals whose offset constitutes approximately one half of the clock cycle, see Figure 4.1 right.

Constructing Master-Slave Flip-Flops and Binary Registers. We introduce a reaction network that mimics a master-slave flip-flop (MSFF) based on the aforementioned chemical clocks and bit manipulating reaction motifs. Moreover, a chain of MSFFs forms a register R_h ($h \in \{1, \ldots, |R|\}$) with bitwise extendable initial length of one bit. In operation, it processes binary numbers $\ldots b_{l_h} b_{l_h-1} \ldots b_2 b_1$ with $b_\alpha \in \{0, 1\}$. Furthermore, each register is equipped with predefined triggers in order to carry out micro-operations "increment", "nonnegative decrement", and "comparison to zero", each of which is processed within one clock cycle.

Within a MSFF, bit setting is coupled to specific edges of the clock signal in order to prevent premature switches. In our MSFF implementation, bit setting consists of two phases (master and slave part). Within the master part, a bit can be preset using specific master species M^T and M^F co-triggered by positive edges of the clock signal C_1, while the subsequent slave part finalises the setting by forwarding the preset bit from the master species to the correlated slave species S^T and S^F triggered by positive edges of the offset clock signal C_2. A subnetwork consisting of eight switching reactions (see darkest grey highlighted boxes within each MSFF in Figure 4.2) covers this task.

With regard to the functionality of a register machine, a sequence of interconnected MSFFs represents a register. Interconnections between neighboured MSFFs reflect the capability of incrementing and decrementing register contents. In case of incrementation, designated trigger molecules INC_h^j effect a successive bit flipping: Starting from the least significant bit b_1, "1" is consecutively converted into "0" until "0" appears first time which is finally converted into "1". Intermediate carry species F_α^I act as forwarding triggers between consecutive bits, see Figure 4.2. If the most significant bit b_{l_h} is reached increasing the concentration of carry species $F_{l_h}^I$, six new species $M_{l_h+1}^T$, $M_{l_h+1}^F$, $S_{l_h+1}^T$, $S_{l_h+1}^F$, $F_{l_h+1}^D$, and $F_{l_h+1}^I$ are added to the reaction system together with the corresponding set of reactions forming the subnetwork for managing bit b_{l_h+1} including update of $M_{l_h+1}^F$ and $M_{l_h+1}^T$ within reactions performing comparison to zero, see Figure 4.2.

Decrementation is organised in a similar way using initial triggers DEC_h^j and intermediate molecules of carry species F_β^D. In order to achieve nonnegative processing, a species E_h^F indicating equality to zero, set by a satellite network, prevents decrementation of binary strings $0 \ldots 0$. Figure 4.2 shows the reaction network structure of a register whose species F_α^I, F_β^D, M_γ^F, M_γ^T, S_γ^F, and S_γ^T are specific with respect to both register identifier h and bit position l_h within the register. Any comparison to zero is done by a satellite network which uses presence of any species M_κ^T with $\kappa = 1, \ldots, l_h$ as triggers in order to flip an equality indicator bit e (species E_h^T and E_h^F) onto "0", while all species M_κ^F with $\kappa = 1, \ldots, l_h$ are needed for flipping onto "1", respectively. The indicator e can be used for program control, see next section. As a further byproduct of each micro-operation on a register, molecules of the form $\#_j \in L$ encoding the jump label of the subsequent machine instruction are released.

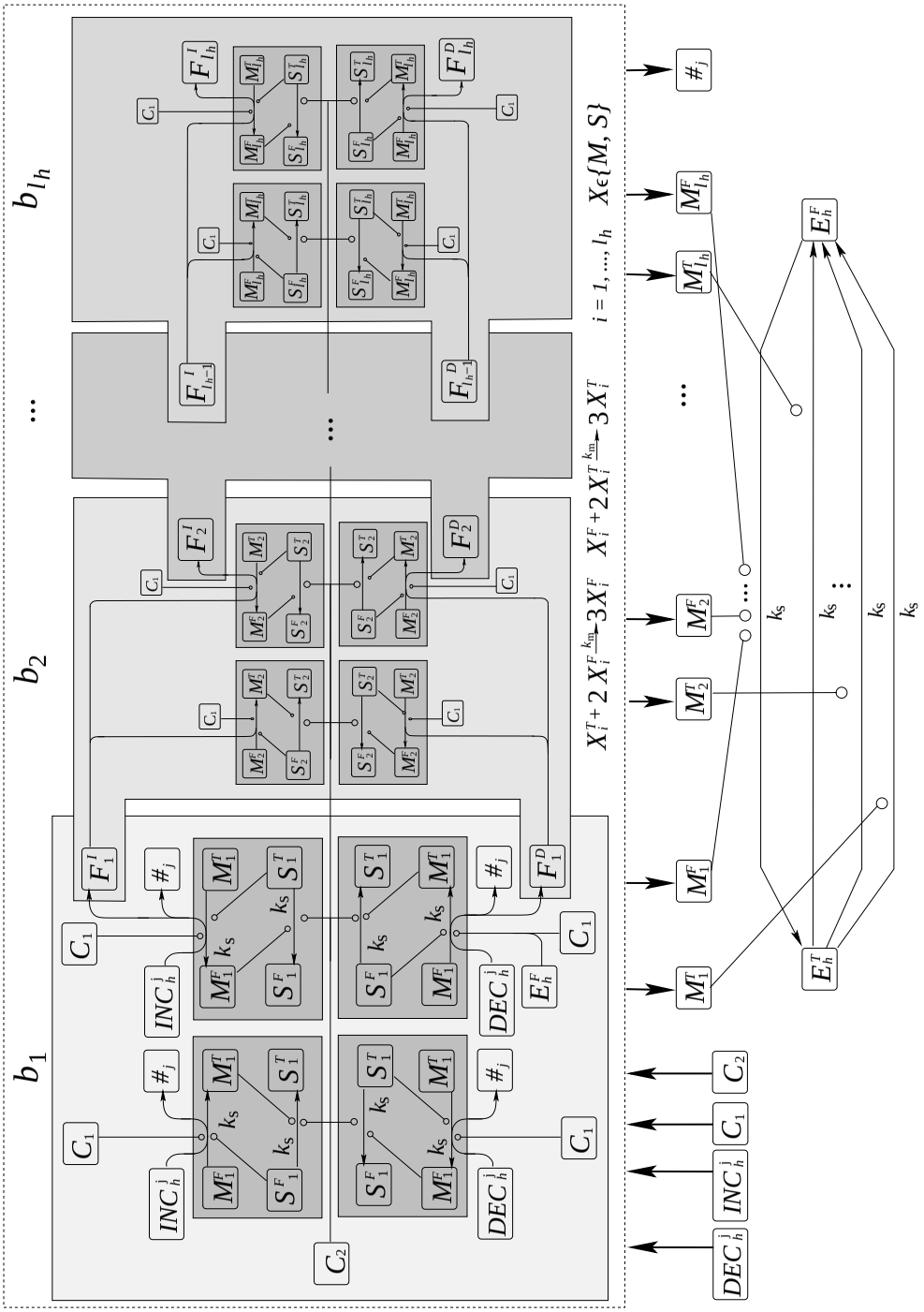


Fig. 4.2 Chemical reaction network of a register capable of processing a bitwise extendable binary number $\ldots b_{l_h} b_{l_h-1} \ldots b_2 b_1$ with $b_\alpha \in \{0,1\}$ including interfaces for micro-operations increment, nonnegative decrement, and comparison to zero.

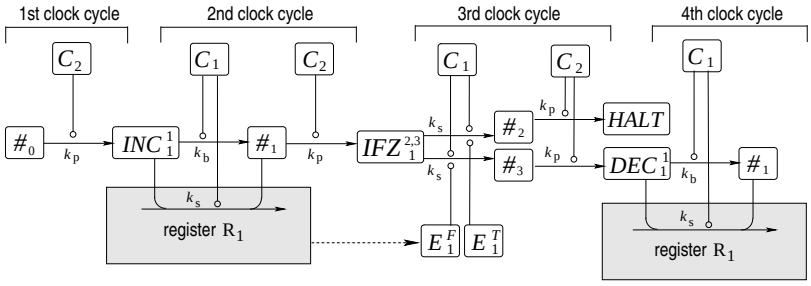


Fig. 4.3 Example of the chemical program control for register machine $M = (\{\mathrm{R}_1\}, \{\#_0, \ldots, \#_3\}, P, \#_0)$ with $P = \{\#_0 : \mathrm{INC}\ \mathrm{R}_1\ \#_1,\quad \#_1 : \mathrm{IFZ}\ \mathrm{R}_1\ \#_2\ \#_3,\quad \#_2 : \mathrm{HALT},\quad \#_3 : \mathrm{DEC}\ \mathrm{R}_1\ \#_1\}$

Implementing a Chemical Program Control. A sequence of reactions directly derived from the given program P of the underlying register machine $M = (R, L, P, \#_0)$ carries out the program control as follows: For each jump label $\#_j \in L$ we introduce a dedicated *label species* $\#_j$ with initial concentrations $[\#_0](0) = 1$ and $[\#_\kappa](0) = 0$ for $\kappa \in \{1, \ldots, |L|-1\}$. Accordingly, a set of *instruction species* $I_\nu \in \{INC_h^j, DEC_h^j \mid \forall h \in \{1, \ldots, |R|\} \wedge \forall j \in \{0, \ldots, |L| - 1\}\} \cup \{IFZ_h^{j,q} \mid \forall h \in \{1, \ldots, |R|\} \wedge \forall j, q \in \{0, \ldots, |L| - 1\}\} \cup \{HALT\}$ is created with initial concentration $[I_\nu](0) = 0$. Furthermore, for each instruction in P a network motif consisting of a *program-control reaction* with kinetic constant $k_\mathrm{p} < k_\mathrm{s}$ and a consecutive *bypass reaction* with $k_\mathrm{b} \leq k_\mathrm{s}$ is defined. Following the two-phase structure of a register machine instruction, these reactions first consume its incipient label species, then produce the corresponding instruction species as an intermediate product and finally convert it into the label species of the subsequent instruction if available. In order to strictly sequentialise the execution of instructions according to the program P, clock species C_1 and C_2 with offset concentration course provided by both oscillators trigger program-control and bypass reactions alternating as catalysts.

The set of reactions for each type of register machine instruction is defined as follows:

instruction	reactions
$\#_i : \mathrm{INC}\ \mathrm{R}_h\ \#_j$	$\#_i + C_2 \xrightarrow{k_\mathrm{p}} INC_h^j + C_2$ $INC_h^j + C_1 \xrightarrow{k_\mathrm{b}} \#_j + C_1$
$\#_i : \mathrm{DEC}\ \mathrm{R}_h\ \#_j$	$\#_i + C_2 \xrightarrow{k_\mathrm{p}} DEC_h^j + C_2$ $DEC_h^j + C_1 \xrightarrow{k_\mathrm{b}} \#_j + C_1$
$\#_i : \mathrm{IFZ}\ \mathrm{R}_h\ \#_j\ \#_q$	$\#_i + C_2 \xrightarrow{k_\mathrm{p}} IFZ_h^{j,q} + C_2$ $IFZ_h^{j,q} + E_h^T + C_1 \xrightarrow{k_\mathrm{s}} \#_j + E_h^T + C_1$ $IFZ_h^{j,q} + E_h^F + C_1 \xrightarrow{k_\mathrm{s}} \#_q + E_h^F + C_1$
$\#_i : \mathrm{HALT}$	$\#_i + C_2 \xrightarrow{k_\mathrm{p}} HALT + C_2$

Instruction species of the form INC_h^j act as triggers for incrementing the contents of register R_h done within its reaction network part, see Figure 4.2. Here, INC_h^j is converted into the byproduct $\#_j$ that provides the label species of the subsequent instruction. Accordingly, species DEC_h^j initiate a set of reactions decrementing register R_h nonnegatively. Instruction species of the form $IFZ_h^{j,q}$ utilise a reaction network module attached to register R_h that releases two species E_h^T and E_h^F whose concentrations indicate whether or not $\mathrm{R}_h = 0$. Instruction species of the form INC_h^j, DEC_h^j, and $IFZ_h^{j,q}$ react into the corresponding label species $\#_j$ and $\#_q$. Since there is no reaction with instruction species $HALT$ as reactant, the program stops in this case. Figure 4.3 illustrates an example of a chemical program control that also gives an overview about the interplay of all predefined modules.

Although instruction species are consumed within register modules, this process could be too slow in a way that a significant concentration of an instruction species outlasts the clock cycle. This unwanted effect is eliminated by bypass reactions running in parallel to the designated register operation.

Case Study: Integer Addition. A chemistry processing $\mathrm{R}_2 := \mathrm{R}_2 + \mathrm{R}_1; \mathrm{R}_1 := 0$ including previous register initialisation $(\mathrm{R}_1, \mathrm{R}_2) := (2, 1)$ on extendable bit word registers emulates a case study of the integer addition "$2 + 1$" whose dynamical behaviour using $k_\mathrm{s} = 3$, $k_\mathrm{m} = 1$, $k_\mathrm{mo} = 3$, $k_\mathrm{b} = 0.5$, $k_\mathrm{p} = 1$ is shown in Figure 4.4 (upper part).

Starting with empty one-bit chemical registers $\mathrm{R}_1 = 0$ and $\mathrm{R}_2 = 0$, the primary P system $\Pi_{\mathrm{PMA}}^{(0)}$ is set into operation. Along with the second incrementation of R_1, concentration of the carry species $F_{1,1}^I$ becomes > 0 initiating the first P system transition into $\Pi_{\mathrm{PMA}}^{(1)}$, see Figure 4.4 (lower part). This system contains additional species and reactions (according to Figure 4.2) to enlarge register R_1 onto two bits. Four C_2 clock cycles later, carry species $F_{2,1}^I$ reaches a positive concentration transforming $\Pi_{\mathrm{PMA}}^{(1)}$ into $\Pi_{\mathrm{PMA}}^{(2)}$ by extending the chemical register R_2 from one into two bit storage capacity.

All simulations of the dynamical register machine behaviour were carried out using CellDesigner version 3.5.2, an open source software package for academic use [9]. The register machine (available from the authors upon request) was implemented in SBML (Systems Biology Markup Language) [11], a file format shown to be suitable for P systems representation [15].

5 Evolutionary Construction of Reaction Networks

Artificial evolution of reaction networks towards a desired dynamical behaviour is a powerful tool to automatically devise complex systems capable of computational tasks. We have designed and implemented a software (SBMLevolver) [12] for evolutionary

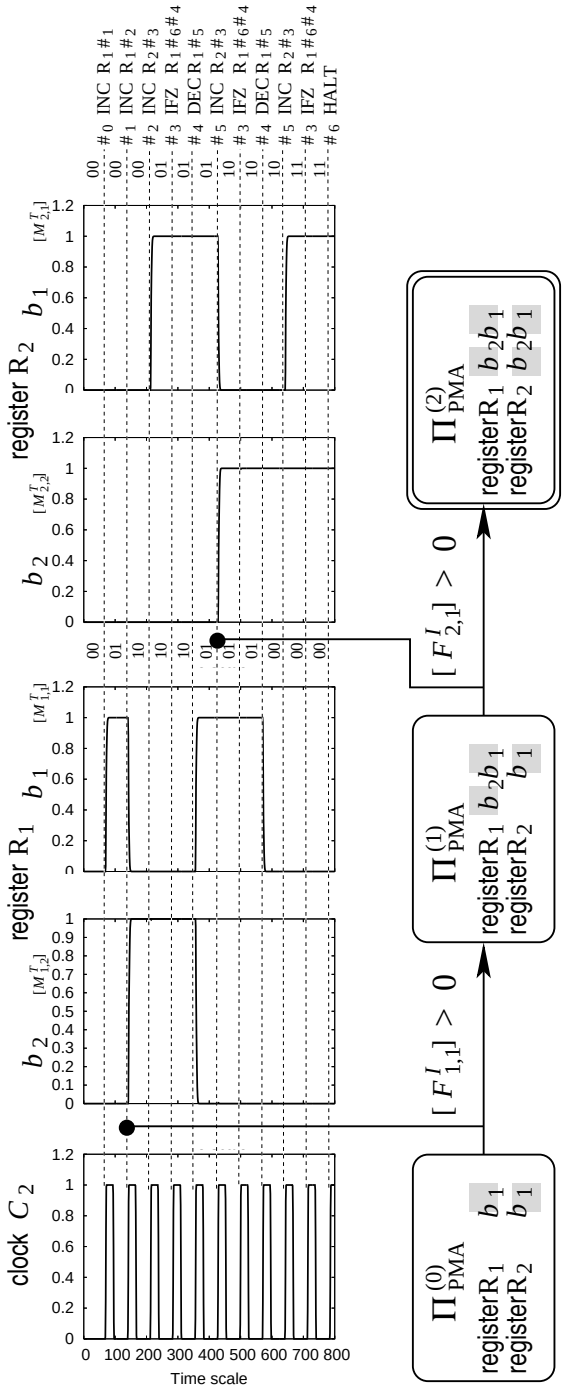


Fig. 4.4 Dynamical behaviour of a chemical register machine acting as an adder (upper part) and transitions between corresponding P systems successive enlarging storage capacity (lower part)

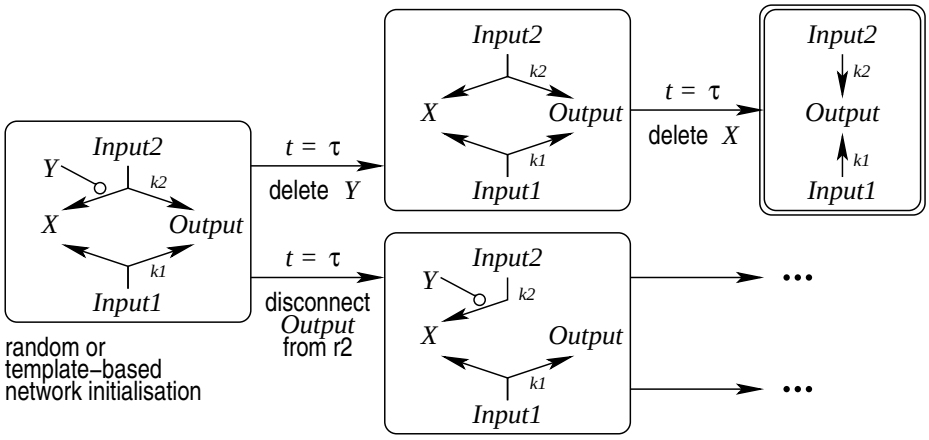


Fig. 5.5 Part of a state transition system sketching the trace of an artificial structural evolution towards a reaction network for addition of two numbers given as initial concentrations of species $Input1$ and $Input2$. In the SBMLevolver, each network passes a separate supplementary parameter fitting (optimisation) of kinetic constants (not shown).

construction of single-compartmental biological models written in SBML. The SBMLevolver enables both, structural evolution (operators: adding/deleting species, adding/deleting reactions, connection/disconnection of a species to/from a reaction, species duplication) and network parameter fitting (adaptation of kinetic constants). Each reaction network generated within the process of artificial evolution forms a P system of the class Π_{PMA}. Evolutionary operators become activated randomly after a dedicated period for running a reaction network. When we understand evolutionary operators as (state) transitions between P systems, the arising phylogenetic graph (history of artificial evolution) is related to the corresponding state transition system. Because state transitions between P systems are not necessarily deterministic, the phylogenetic graph may have multiple branches. An example in Figure 5.5 shows P system transitions sketching an artificial evolution process towards a reaction network for addition of two numbers. In this procedure, selection can be incorporated by a network evaluation measure to be included as a component of Π_{PMA}.

6 Conclusions

Formalisation of complex biological or chemical systems with structural dynamics within their reaction rules can contribute to explore the potential of their functionality as a whole. From the modelling point of view, coordination of temporally local subsystem descriptions in terms of well-defined interfaces might be a challenging task since it requires a homogeneous approach. The P systems framework inherently suits here because of its discrete manner and its ability to combine different levels of abstraction.

We have shown a first idea for arranging previously separate subsystems into a common temporal framework. In our approach, transitions between subsystems have been initiated by constraints denoted as boolean expressions. Therefore, we allow for evaluation of internal signals (molecular amount) as well as external signals (time provided by a global clock). Beyond computational completeness, application scenarios are seen in systems and synthetic biology. Further work will be directed to comprise P systems of different classes and with compartmental structures into a common transition framework.

Acknowledgements. This work is part of the ESIGNET project (Evolving Cell Signalling Networks *in silico*), which has received research funding from the European Community's Sixth Framework Programme (project no. 12789). Further funding from the German Research Foundation (DFG, grant DI852/4-2) is gratefully acknowledged.

Bibliography

[1] U. Alon. *An Introduction to Systems Biology.* Chapman & Hall, 2006
[2] B.P. Belousov. A Periodic Reaction and Its Mechanism. *Compilation of Abstracts in Radiation Medicine* **147**:145, 1959
[3] R.E. Blankenship. *Molecular Mechanisms of Photosynthesis.* Blackwell Science, 2002
[4] H.M. Brody et al. *Phenotypic Plasticity.* Oxford University Press, 2003
[5] K.A. Connors. *Chemical Kinetics.* VCH Publishers, 1990
[6] D. Debanne. *Brain plasticity and ion channels. Journal of Physiology* 97(4-6):403-414, 2003
[7] P. Dittrich et al. Artificial Chemistries: A Review.*Artificial Life* **7(3)**:225-275, 2001
[8] F. Fontana et al. Discrete Solutions to Differential Equations by Metabolic P Systems. *Theor. Comput. Sci.* **372(1)**:165-182, 2007
[9] A. Funahashi et al. CellDesigner: a process diagram editor for gene-regulatory and biochemical networks. *Biosilico* **1**:159-162, 2003 (`www.celldesigner.org`)
[10] T. Hinze et al. Hill Kinetics Meets P Systems. In G. Eleftherakis et al. (Eds.) *Membrane Computing. LNCS* **4860**:320-335, Springer Verlag, 2007
[11] M. Hucka et al. The systems biology markup language SBML: A medium for representation and exchange of biochemical network models. *Bioinformatics* **19(4)**: 524-531, 2003
[12] T. Lenser et al. Towards Evolutionary Network Reconstruction Tools for Systems Biology. In E. Marchiori et al. (Eds.) *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. LNCS* **4447**:132-142, Springer Verlag, 2007
[13] M.O. Magnasco. Chemical Kinetics is Turing Universal. *Physical Review Letters* **78(6)**:1190-1193, 1997

[14] V. Manca. Metabolic P Systems for Biomolecular Dynamics. *Progress in Natural Sciences* **17(4)**:384-391, 2006

[15] I. Nepomuceno et al. A tool for using the SBML format to represent P systems which model biological reaction networks. *Proceedings 3rd Brainstorming Week on Membrane Computing.* pp. 219-228, 2005

[16] G. Păun. Computing with Membranes. *J.Comp.Syst.Sci.* **61(1)**:108-143, 2000

[17] G. Păun. *Membrane Computing: An Introduction.* Springer Verlag Berlin 2002

[18] D. Pescini et al. Investigating local evolutions in dynamical probabilistic P systems. In G. Ciobanu et al. (Eds.) *Proceedings First International Workshop on Theory and Application of P Systems.* pp. 275-288, 2005

[19] G. Rozenberg, A. Salomaa (Eds.) *Handbook of Formal Languages.* Vol. I-III, Springer Verlag Berlin, 1997

[20] Y. Suzuki, H. Tanaka. Symbolic chemical system based on abstract rewriting system and its behavior pattern. *Artificial Life and Robotics* **1(4)**:211-219, 1997

[21] C.G.N. Mascie-Taylor, B. Bogin (Eds.) *Human Variability and Plasticity.* Cambridge University Press, 1995

[22] A.M. Zhabotinsky. Periodic Processes of Malonic Acid Oxidation in a Liquid Phase. *Biofizika* **9**:306-311, 1964

Effects of HIV-1 Proteins on the Fas-Mediated Apoptotic Signaling Cascade: A Computational Study of Latent CD4+ T Cell Activation

John Jack[1], Andrei Păun[1,2,3], Alfonso Rodríguez-Patón[2]

[1]Louisiana Tech University, Department of Computer Science/IfM
P.O. Box 10348, Ruston, LA 71272, USA
{johnjack, apaun}@latech.edu

[2]Universidad Politécnica de Madrid, Departamento de Inteligencia Artificial,
Facultad de Informática
Campus de Montegancedo S/N, Boadilla del Monte, 28660 Madrid, Spain
arpaton@fi.upm.es

[3]National Institute of Research and Development for Biological Sciences,
Bioinformatics Department,
Splaiul Independenţei, Nr. 296, Sector 6, Bucharest, Romania

We present a new model for simulating Fas-induced apoptosis in HIV-1-infected CD4+ T cells. Moreover, the reactivation of latently infected cells is explored. The work, an extension of our previous modeling efforts, is the first attempt in systems biology for modeling the Fas pathway in the latently infected cells. These enigmatic cells are considered the last barrier in the elimination of HIV infection. In building the model, we gathered what reaction rates and initial conditions are available from the literature. For the unknown constants, we fit the model to the available information on the observed effects of HIV-1 proteins in activated CD4+ T cells. We provide results, using the Nondeterministic Waiting Time (NWT) algorithm, from the model, simulating the infection of activated CD4+ T cells as well as the reactivation of a latently infected cells. These two model versions are distinct with respect to the initial conditions – multiplicities and locations of proteins at the beginning of the simulation.

1 Introduction

1.1 Motivation for Study The human immunodeficiency virus (HIV) is remarkable for several reasons: (1) it predominantly **infects immune system cells**; (2) shows a **high genetic variation** throughout the infection in a single individual due to the high error rate in the reverse transcription; (3) it **induces apoptosis**, or cellular suicide, in the "healthy" (bystander) immune cells; and (4) normal immune system function can cause some HIV-infected T cells to become **latent**, entering a reversibly nonproductive

state of infection. Since the latent cells are transcriptionally silent, they are virtually indistinguishable from the uninfected cells. Also, the number of latently infected cells is relatively small, around 3% of the T cells, which makes the experimental study of these cells difficult – current technology in biochemistry requires large numbers of the molecules/cells to be studied. It is widely believed that the latently infected CD4+ T cells represent the last barrier to an HIV cure. The current paper represents an initial modeling effort for the apoptosis (programmed cell death) of latently infected T cells.

We will focus on the apoptotic modeling (reason 3), since it is the avenue through which the virus destroys the effectiveness of the host's immune system. We will base our model on the previous modeling work of [21], using the simulation technique reported in [23]. Furthermore, in order to make the modeling effort easier and due to the high genetic variability (reason 2) of the viral genome, we will combine several similar processes together into single reactions. The kinetic constants for the new reactions, modeling the biochemical interactions involving viral proteins with the host cell, will be obtained by fitting the model to reported experiments on the infected, nonlatent cells. Finally, we will simulate the latent cells (immediately after they are reactivated) by adjusting the appropriate initial conditions of the system.

1.2 AIDS Pathogenesis As far as we know, this paper reports the first attempt at modeling the Fas-mediated apoptotic signaling pathway in reactivated latently infected CD4+ T cells. Although there are two strains of HIV, type 1 and type 2, we are interested in HIV-1, since it is more virulent and transmissive [37]. HIV-1 is called a global pandemic by the World Health Organization (WHO). Since its discovery over two decades ago, the virus has been the target of aggressive research. And yet, a cure – complete eradication of the viral infection – remains out of reach. According to statistics from the WHO, there were 33.2 million people living with HIV in 2007, 2.5 million newly infected individuals, and 2.1 million AIDS deaths [47].

The pathogenesis of AIDS is attributed to the depletion of the host's CD4+ T cells, the loss of which results in a dysfunctional immune system. Finkel et al. in [15] concluded that HIV-1 infection causes death predominantly in bystander T cells. These healthy, uninfected cells are marked for destruction by the neighboring HIV-1-infected cells. The mechanism of the bystander cell death was shown to be apoptosis. Proteins encoded by the HIV-1 genome exhibit anti- and pro-apoptotic behavior on infected and bystander cells, enhancing or inhibiting a cell's ability to undergo apoptosis. There are numerous drugs available for limiting the impact of HIV-1 on the immune system; the most successful approach, highly active anti-retroviral therapy (HAART), is a combination of several types of drugs, targeting different mechanisms of HIV-1 infection and proliferation.

Although HAART has proven to be effective in the reduction or elimination of viremia [34], it is ineffective in the complete eradication of the viral infection. Latent reservoirs of HIV-1 have been detected in HIV-1-infected patients [9, 10]. Latently infected cells

are relatively rare – about 1 in 10^6 resting T cells [10]. However, they are considered to be the largest obstacle in combating HIV-1 infection [16, 42, 44]. Understanding the mechanisms behind HIV-1 latency is a focal point for current AIDS-related research (for a recent review on latency see [19]).

There are two types of latency described in the literature. The first, preintegration latency, refers to resting T cells containing unintegrated HIV-1 DNA. Since the unintegrated HIV-1 DNA is labile and reverse transcription of HIV-1 RNA is slow (on the order of days) [35, 50, 51, 53], it is believed that patients with reduced viremia after several months of HAART therapy do not have resting T cells with unintegrated HIV-1 DNA [7]. However, resting T cells with stably integrated HIV-1 DNA can provide a reservoir for viral reproduction for years [16]. These reservoirs are the result of activated HIV-1-infected T cells that have returned to a quiescent state. Due to their long lifespan, we have chosen to model the apoptotic events that follow the reactivation of a postintegration latently infected CD4+ T cell. N.B., for the remainder of the paper, when use the term latent, we are referring to the postintegration latency.

We have previously reported our results [23] from simulating the Fas-mediated apoptotic signaling cascade, based on information for the Jurkat T cell line [21]. In [23], we provided an exhaustive study on the feasibility of our Nondeterministic Waiting Time (NWT) algorithm, comparing our results to an established ordinary differential equations (ODEs) technique [21]. We have extended the Fas model, incorporating the effects HIV-1 proteins have on the pathway.

In Section 2, we provide a brief summary of our simulation technique. Section 3 discusses the background information on the Fas pathway and HIV-1 proteins necessary to understanding our model. Section 4 contains the results of our simulations. Finally, Section 5 is a discussion of issues revolving around modeling HIV-1 protein activity and future research plans for our group.

2 The NWT Algorithm

We refer the interested reader to [23], where we gave a detailed description of the NWT algorithm. We will now highlight the key aspects of our simulation technique.

The NWT algorithm is a Membrane Systems implementation, where the alphabet of the system is defined as proteins, and the rules are the reactions involving the proteins. The algorithm is mesoscopic, since individual molecules are employed instead of molecular concentrations. This allows us to discretely interpret the evolution of the intracellular molecular dynamics. We have argued in [23] that our discrete, nondeterministic algorithm may outperform other continuous methods – for example, ODE simulations – in situations of low molecular multiplicity.

All of the reactions within the system obey the Law of Mass Action – i.e., the amount

of time required for any given reaction to occur is directly proportional to the number of reactant molecules present in the system. The Law of Mass Action is used to calculate a waiting time (WT) for each reaction, indicating the next occurrence of the reaction. These values are based on kinetic constants and are deterministic – the nondeterminism in our algorithm stems from reaction competition over low molecular multiplicity. Our NWT algorithm is different than the Gillespie algorithm [17, 18], where stochastic values are generated to govern the reaction rates. We will use the NWT algorithm to explore the effects of HIV-1 proteins on the Fas-mediated signaling cascade.

3 The Model

3.3 Fas-Mediated Apoptosis We have explored the literature pertaining to the effects of HIV-1 proteins on apoptosis: see [2, 38, 41] for reviews on HIV-1-related CD4+ T cell death. There are several distinct death receptors, which, upon activation of the cell, can lead to cellular apoptosis through a tightly regulated molecular signaling cascade [3]. In this paper, our concern is the Fas pathway. As reported in [22] and [36], understanding the complex signaling cascade of Fas-mediated apoptosis can be beneficial in developing remedies for cancer and autoimmune disorders.

In [40], the authors describe two signaling pathways for Fas-mediated apoptosis: type I and type II. Both pathways begin with the Fas ligand binding to the Fas receptor (also called CD95) on the cell membrane. This results in a conformational change at the receptor, producing a complex, Fasc. The cytoplasmic domain of this complex recruits Fas-associated death domain (FADD). A maximum of three FADD molecules can be recruited to each Fasc molecule. Once FADD is bound to Fasc, Caspase 8 and FLIP are recruited competitively. Although three molecules of Caspase 8 can be recruited to each Fasc-FADD binding, only two are required to create the dimer, Caspase $8_2^{P_{41}}$, which is released into the cytoplasm. The cytoplasmic Caspase $8_2^{P_{41}}$ is then phosphorylated into active form (Caspase 8*). The binding of FLIP to Fasc inhibits apoptosis, because it reduces the ability of Caspase 8 to become activated – i.e., FLIP can occupy binding sites necessary for creation of Caspase $8_2^{P_{41}}$.

Unless sufficiently inhibited, the Fas signaling cascade continues in the type I or type II pathway. For sufficiently large Caspase 8 initial concentration, Caspase 3 is directly phosphorylated by the Caspase 8*. This is the type I pathway. If the number of Caspase 8 molecules is insufficient to induce Caspase 3 activation directly, then the type II pathway is initiated. Caspase 8* truncates the Bid protein, tBid. The tBid protein can then bind to two molecules of Bax. The complex formed by this binding leads to the release of Cytochrome c from the mitochondria. Once it is translocated to the cytoplasm, Cytochrome c binds to Apaf and ATP, forming a complex that can recruit and activate Caspase 9 (Caspase 9*). The activated Caspase 9* proceeds to activate Caspase 3. We consider the activation of Caspase 3 to be the end of the signaling cascade, since its active form signals DNA fragmentation [28].

In [23], we modeled both the type I and type II Fas-induced apoptotic signaling pathways. In the next section, we discuss HIV-1 infection and its effects on the Fas signaling cascade.

3.4 HIV-1 Infection The mechanisms behind HIV-1 infection of CD4+ T cells are well understood. A spike on the virus, the gp120 envelope glycoprotein, binds to the CD4 receptor of the target cell and, in conjunction with subsequent binding to a coreceptor (CCR5 or CXCR4), a path is opened for the virus to inject its contents into the cell [8, 48]. Reverse transcriptase creates cDNA from the HIV-1 RNA and the genome of the virus is implanted into the cell's own DNA for future production. During this time, the immune system fails to detect and destroy the infected cell.

There is still some debate about the effects of HIV-1 proteins on cellular signaling networks; however, we have pooled the collective knowledge of the biological community in order to categorize and model the described functions of various HIV proteins. For an illustration of the Fas pathway and the involvement of the HIV proteins we refer the reader to Fig. 3.1.

Upon infection, the contents of the virion (e.g., Vpr, HIV protease (HIV_{pr}), reverse transcriptase (RT), and HIV RNA (HIV_{RNA})) are released into the cytoplasm [6]. In the newly infected, activated CD4+ T cell the HIV_{RNA} is converted to cDNA (HIV_{cDNA}) by the reverse transcriptase about five hours post-infection [25]. The HIV_{cDNA} is then integrated into the host's genome with the help of the viral integrase approximately one hour later [14]. These rules are shown in Table 3.1. For our convenience, we have labeled the integrated HIV genome as HIV_{LTR} in our rules. HIV_{LTR} is the basis for interactions involving the HIV long terminal repeat; in our model, it is a necessary component for all reactions pertaining to HIV-1 protein production.

After integration of the viral DNA, gene expression of HIV proteins becomes possible. The nuclear factor of activated T cells (NFAT) and NF-κB have been shown to play important roles in HIV gene expression [26, 30]. In a resting CD4+ T cell, NF-κB is sequestered in the cytoplasm by its inhibitor, IκB. Following cellular activation, NF-κB is released by its inhibitor, which allows it to relocate to the nucleus where it can bind to the HIV_{LTR}. Also following T cell activation, NFAT, located in the cytoplasm of resting CD4+ T cells, undergoes dephosphorylation and translocation to the nucleus where it can bind to the HIV_{LTR} [26]. Once NF-κB and NFAT are translocated to the nucleus, they can bind to the HIV_{LTR}, combining their efforts to synergistically enhance the promoter activity. Moreover, [26] shows that the combined effects of Tat, NF-κB and NFAT is much stronger than the pairings of Tat and NF-κB or Tat and NFAT. In our model, we have combined the roles of NF-κB and NFAT. Hence, the translocation and binding rules for NFAT (and NF-κB) are shown in Table 3.1.

Multiply spliced (MS) HIV-1 mRNAs – responsible for Tat/Rev protein creation – are detectable in resting CD4+ T cells [27]. However, due to the inefficient export of the

Table 3.1 A list of the reactions involving effects of HIV-1 proteins, which were added to the existing Fas model [21, 23]. See Appendix A for the complete list of reactions.

Reaction	**Reaction Rate**
1: HIV_{RNA} + RT $\rightarrow$ HIV_{cDNA} + RT	k_{21}
2: HIV_{cDNA} $\rightarrow$ HIV_{cDNA} (nuclear import)	k_{22}
3: HIV_{cDNA} $\rightarrow$ HIV_{LTR}	k_{22}
4: NFAT $\rightarrow$ NFAT (nuclear import)	k_{23}
5: CDK9 $\rightarrow$ CDK9 (nuclear import)	k_{24}
6: CyclinT1 + CDK9 $\rightarrow$ PTEFb	k_{25}
7: NFAT + HIV_{LTR} $\rightarrow$ HIV_{LTR}:NFAT	k_{26}
8: HIV_{LTR}:NFAT + Tat $\rightarrow$ HIV_{LTR}:NFAT:Tat	k_{27}
9: HIV_{LTR}:NFAT:Tat + PTEFb $\rightarrow$ HIV_{LTR}:NFAT:Tat:PTEFb	k_{28}
10: HIV_{LTR} $\rightarrow$ HIV_{LTR} + $mRNA_{Tat}$	k_{29}
11: HIV_{LTR} $\rightarrow$ HIV_{LTR} + $mRNA_{Vpr}$	k_{29}
12: HIV_{LTR} $\rightarrow$ HIV_{LTR} + $mRNA_{HIV_{pr}}$	k_{29}
13: HIV_{LTR} $\rightarrow$ HIV_{LTR} + $mRNA_{Nef}$	k_{29}
14: HIV_{LTR}:NFAT $\rightarrow$ HIV_{LTR}:NFAT + $mRNA_{Tat}$	k_{30}
15: HIV_{LTR}:NFAT $\rightarrow$ HIV_{LTR}:NFAT + $mRNA_{Vpr}$	k_{30}
16: HIV_{LTR}:NFAT $\rightarrow$ HIV_{LTR}:NFAT + $mRNA_{HIV_{pr}}$	k_{30}
17: HIV_{LTR}:NFAT $\rightarrow$ HIV_{LTR}:NFAT + $mRNA_{Nef}$	k_{30}
18: HIV_{LTR}:NFAT:Tat $\rightarrow$ HIV_{LTR}:NFAT:Tat + $mRNA_{Tat}$	k_{31}
19: HIV_{LTR}:NFAT:Tat $\rightarrow$ HIV_{LTR}:NFAT:Tat + $mRNA_{Vpr}$	k_{31}
20: HIV_{LTR}:NFAT:Tat $\rightarrow$ HIV_{LTR}:NFAT:Tat + $mRNA_{HIV_{pr}}$	k_{31}
21: HIV_{LTR}:NFAT:Tat $\rightarrow$ HIV_{LTR}:NFAT:Tat + $mRNA_{Nef}$	k_{31}
22: HIV_{LTR}:NFAT:Tat:PTEFb $\rightarrow$ HIV_{LTR}:NFAT:Tat:PTEFb + $mRNA_{Tat}$	k_{32}
23: HIV_{LTR}:NFAT:Tat:PTEFb $\rightarrow$ HIV_{LTR}:NFAT:Tat:PTEFb + $mRNA_{Vpr}$	k_{32}
24: HIV_{LTR}:NFAT:Tat:PTEFb $\rightarrow$ HIV_{LTR}:NFAT:Tat:PTEFb + $mRNA_{HIV_{pr}}$	k_{32}
25: HIV_{LTR}:NFAT:Tat:PTEFb $\rightarrow$ HIV_{LTR}:NFAT:Tat:PTEFb + $mRNA_{Nef}$	k_{32}
26: $mRNA_{Tat}$ $\rightarrow$ $mRNA_{Tat}$ (nuclear export)	k_{33}
27: $mRNA_{Nef}$ $\rightarrow$ $mRNA_{Nef}$ (nuclear export)	k_{33}
28: $mRNA_{Vpr}$ $\rightarrow$ $mRNA_{Vpr}$ (nuclear export)	k_{33}
29: $mRNA_{HIV_{pr}}$ $\rightarrow$ $mRNA_{HIV_{pr}}$ (nuclear export)	k_{33}
30: $mRNA_{Tat}$ $\rightarrow$ $mRNA_{Tat}$ + Tat	k_{34}
31: $mRNA_{Nef}$ $\rightarrow$ $mRNA_{Nef}$ + Nef	k_{34}
32: $mRNA_{Vpr}$ $\rightarrow$ $mRNA_{Vpr}$ + Vpr	k_{34}
33: $mRNA_{HIV_{pr}}$ $\rightarrow$ $mRNA_{HIV_{pr}}$ + HIV_{pr}	k_{34}
34: $mRNA_{Tat}$ $\rightarrow$ degraded	k_{35}
35: $mRNA_{Nef}$ $\rightarrow$ degraded	k_{35}
36: $mRNA_{Vpr}$ $\rightarrow$ degraded	k_{35}
37: $mRNA_{HIV_{pr}}$ $\rightarrow$ degraded	k_{35}
38: Tat $\rightleftharpoons$ Tat (nuclear import/export)	k_{36f}, k_{35r}

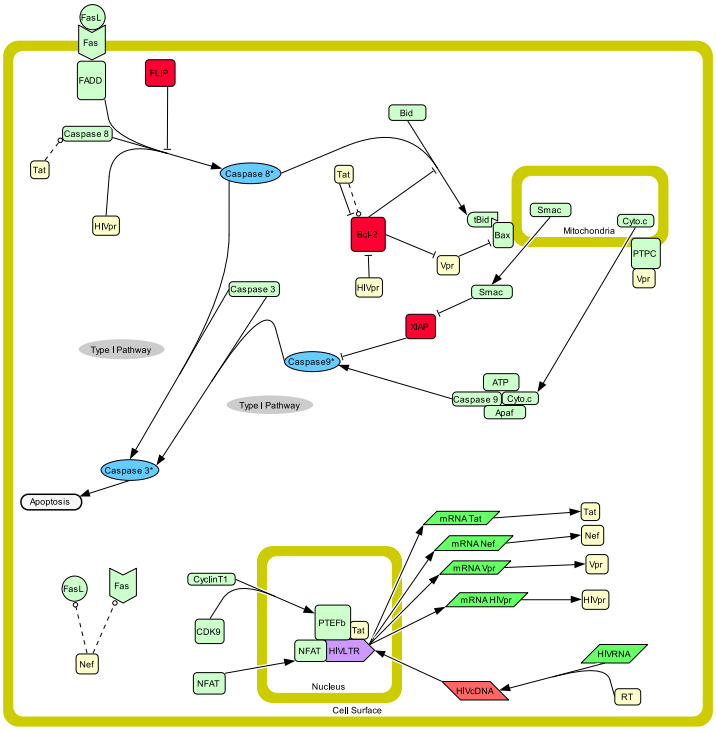


Fig. 3.1 A picture of the model for HIV-1 protein effects on Fas signaling. The activation of Caspase 3 is the end of the signaling cascade – irrevocably leads to cell death. The type I pathway involves direct activation of Caspase 3 by Caspase 8*. The type II pathway requires signal amplification by way of the mitochondria, resulting in the activation of Caspase 3 by Caspase 9*. The HIV-1 Tat protein upregulates inactive Caspase 8 and Bcl-2, but it can also downregulate Bcl-2. Vpr upregulates Bcl-2 and downregulates Bax. HIV Protease can cleave Bcl-2 into an inactive form and it can also cleave Caspase 8 into active Caspase 8. Finally, Nef protein upregulates Fas ligand and Fas receptor.

mRNA transcripts to the cytosol, Tat and Rev proteins are undetectable in the latent cells. Activation of these latent cells leads to production of Tat and Rev, and subsequent upregulation of all HIV-1 proteins. In order for the infected cells to create HIV proteins other than Tat and Rev, the transcriptional elongation induced by Tat and the efficient nuclear export of MS HIV-1 RNAs by Rev are required. Our latent cell model, beginning with cellular activation, initially allows for inefficient creation of Tat proteins. We chose not to model Rev, since it has no known Fas apoptotic function; its exporting functions are incorporated into the kinetic constants governing mRNA translocation. Once Tat is located in the nucleus, it requires the help of two other proteins provided by the host cell: CyclinT1 and CDK9.

39: Tat → Tat + Casp8	k_{37}
40: Tat → Tat + Bcl2	k_{38}
41: Tat → FasL + Tat	k_{39}
42: Tat + Bcl2 → Tat	k_{40}
43: Vpr + Bax → Vpr	k_{42}
44: Vpr + Bcl2 → Vpr:Bcl2	k_{43}
45: Vpr:Bcl2 → Vpr + Bcl2	k_{44}
46: Vpr + PTPC → Vpr:PTPC	k_{45}
47: Vpr:PTPC + Cyto.c → Cyto.c*+ Vpr:PTPC	k_{46}
48: HIV$_{pr}$ + Casp8 → HIV$_{pr}$ + Casp8*	k_{47}
49: HIV$_{pr}$ + Bcl2 → HIV$_{pr}$	k_{48}
50: Nef → Nef + Fas	k_{49}
51: Nef → Nef + FasL	k_{50}
52: FasL → FasL (to cell surface)	k_{51}

In an inactivated cell, CyclinT1 and CDK9 are sequestered in the cytoplasm [31]. Upon T cell activation, they are relocated to the nucleus. CyclinT1 and CDK9 combine to make up the positive-acting transcription elongation factor (P-TEFb) complex. The binding of P-TEFb and Tat at the HIV$_{LTR}$ allows the hyperphosphorylation of RNA polymerase II (RNAPII), resulting in increased transcriptional elongation. The translocation and binding rules for CyclinT1, CDK9 and Tat are formalized in Table 3.1. The transcription, translocation, and translation rules involving HIV-1 mRNA molecules are also summarized in Table 3.1.

3.5 HIV-1-Related Effects on the Fas Pathway Aside from its role in transcriptional elongation, the Tat protein is responsible for both pro- and anti-apoptotic behavior. In [4], the authors demonstrated that increased Tat expression causes upregulation of inactive Caspase 8. Also, Tat has been associated with the downregulation of Bcl-2 [41]. Given the pro- and anti-apoptotic duties of Caspase 8 and Bcl-2, respectively, it appears that a cell with high levels of Tat has increased susceptibility to apoptosis. Conversely, [15] claims that Tat upregulates Bcl-2, resulting in decreased apoptotic rates of cells. Tat has also been implicated in the upregulation of Fas ligand on the cell surface [4, 49], which may effect the cell through autocrine signaling. The anti- and pro-apoptotic rules for Tat are found in Table 3.1.

The HIV-1 Vpr has been shown to both enhance and inhibit the Fas signaling cascade. Upon infection, the $\sim$700 molecules of Vpr in the virion are injected into the cytoplasm of the cell [6]. At low levels, Vpr has been shown to prohibit apoptosis by upregulating Bcl-2 and downregulating Bax [12]. However, higher concentrations of Vpr affects the mitochondrial membrane permeability via interactions with the permeability transition pore complex (PTPC), resulting in the release of Cytochrome c into the cytoplasm [24].

In the same paper, the authors also demonstrated that Bcl-2 can inhibit the effects of Vpr on the PTPC. The various apoptotic roles of Vpr we define in Table 3.1.

Another protein packaged in HIV-1 virions, HIV_{pr}, plays an important role in the Fas pathway. The HIV_{pr} has been shown to cleave Bcl-2 into a deactivated state [43], while it also cleaves Caspase 8 [32] into active form. Both rules are pro-apoptotic and are in Table 3.1.

Finally, we define two pro-apoptotic rules for the Nef protein. Zauli et al. discovered in [52] that Nef can play a role in cell death by upregulating Fas receptor and Fas ligand on the cell surface. Upregulating the receptor sites of Fas on the cell surface prepares the cell for ligand binding, and can initiate the Fas-induced apoptotic signaling cascade. The upregulation of Fas ligand may protect the infected cell from cytotoxic T cells, or it could be part of autocrinic signaling. The four rules for upregulation and translocation of Fas and Fas ligand are in Table 3.1.

4 Results

We added all of the rules from Table 3.1 to the Fas model described in [21, 23] – see Appendix A for the complete list. From this, we are able to simulate two types of cells: *nonlatent* and *latent*. The differences between the two models are the initial protein multiplicities. The *nonlatent* cell is an activated T cell which has just been infected with the contents of the HIV-1 virion. The HIV-1 RNA and other viral proteins are in the cytoplasm. The HIV-1 RNA must be incorporated into the host's genome before the viral protein production begins. The *latent* model is a newly activated T cell with no HIV-1 proteins present. However, the HIV-1 genome is already integrated into the host's DNA.

As we have discussed earlier, the *nonlatent* cell is used for the model fitting, since the majority of information about HIV-1 proteins pertains to these types of cells. For instance, in Fig. 4.2(a), the results from the *nonlatent* simulation show the activity of Tat in that full length (inactive) Caspase 8 increases by a factor of three. Our simulation agrees with the observations of [4]. Also, in Fig. 4.2(b), our model shows Vpr-induced upregulation of Bcl-2 and downregulation of Bax by 30% and 20%, resp. Our results agree with the experimental results on Vpr described in [12].

We will next consider the activation of Caspase 3. In Fig. 4.3, both the *nonlatent* and *latent* models are shown to exhibit the onset of apoptosis – total activation of Caspase 3 – after approximately two days. Our results indicate that reactivated latently infected CD4+ T cells activate all of the Caspase 3 molecules earlier than the *nonlatent* model. Also, in Fig. 4.3, we show the truncation of Bid, which is a necessary step in the induction of the type II pathway. Active Caspase 8 is responsible for the truncation of Bid, so we are seeing the downstream effects of Caspase 8 activation.

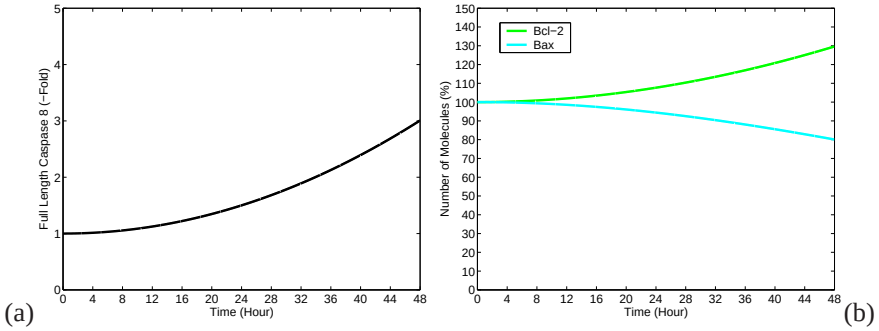


Fig. 4.2 (a) Tat protein upregulates Caspase 8 levels by three-fold. (b) Vpr upregulates Bcl-2 and downregulates Bax by 30% and 20%, resp.

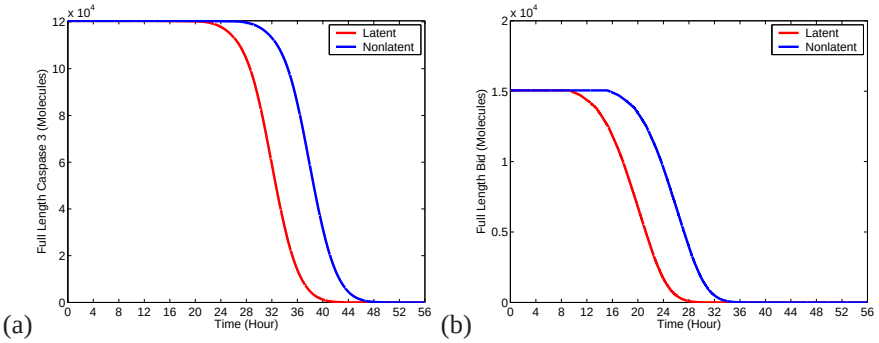


Fig. 4.3 (a)Total reduction of full length Caspase 3 is seen after ~ 40 hours in the *latent* model, whereas the *nonlatent* model takes ~ 47 hours. (b) The decline of Bid through interactions with Caspase 8, leading to a rise in tBid.

Next, let us consider the mechanisms behind Caspase 3 activation in the *latent* and *nonlatent* models. According to the rules in Appendix A, an interaction between full length Caspase 3 and active Caspase 8 or Caspase 9 can have two outcomes: the activation of Caspase 3 or not. Both of our models show cooperation between the two pathways, which is not explicitly stated in the literature. The *nonlatent* results (Fig. 4.4) show the first interactions between Caspase 3 and Caspase 8* molecules occur just after 18 hours into the run. It isn't until ~ 10 hours later (26 hours into the run) that we begin to see Caspase 3 interactions with Caspase 9*, after signal amplification through the mitochondria. As discussed in [21, 23], given a sufficiently high initial concentration of Caspase 8 in the cell, signal amplification is not necessary to induce apoptosis. For this model, we set the initial level of Caspase 8 to be insufficient for apoptosis by the type I pathway.

The results of the *latent* simulation are similar to the *nonlatent*, where both pathways appear to govern Caspase 3 activation. In the *latent* run (Fig. 4.5), we see type I inter-

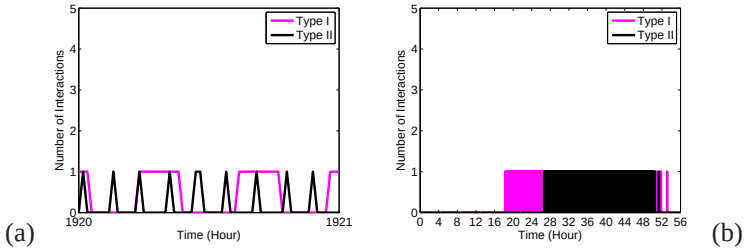


Fig. 4.4 These graphs show the type I and type II pathways working together to activate Caspase 3 during the *nonlatent* simulation. The type I interactions are active Caspase 8 binding with Caspase 3, and the type II interactions are Caspase 9 binding with Caspase 3. (a) The overall picture for the whole three days of simulation. (b) One minute from the simulation (from 32 hours to 32 hours and 1 minute) illustrates the rate of interactions.

actions first occur about 12 hours into the simulation, while type II molecular binding occurs after 21 hours.

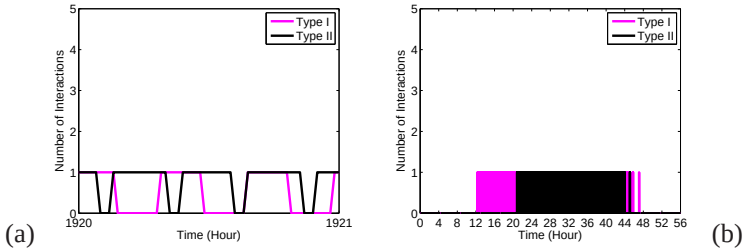


Fig. 4.5 These graphs show the type I and type II pathways working together to activate Caspase 3 during the *latent* simulation. They are similar to the *nonlatent* run. (a) The overall picture for the whole three days of simulation. (b) One minute from the simulation (from 32 hours to 32 hours and 1 minute) illustrates the rate of interactions.

Although Fig. 4.4(b) and Fig. 4.5(b) imply type I interactions occur more frequently than type II, it must be noted that, due to the kinetics governing these binding rules, Caspase 8* can remain bound to Caspase 3 for a longer period of time than Caspase 9*. Therefore, although it seems that Caspase 8* binds to Caspase 3 more frequently, the reactions are merely slower. In fact, both models exhibit more interactions between Caspase9* and Caspase 3.

5 Discussion

Based on the biological evidence in the literature, we constructed a model for the effects of HIV-1 proteins on the Fas-mediated apoptosis pathway. This work is the first of its kind, simulating Fas-induced apoptosis in reactivated latently infected CD4+ T cells. We

have provided some preliminary results in an effort to understand CD4+ T cell latency. Interestingly, our results show a cooperation between the type I and type II pathways. We have not been able to verify an explanation for this in the available literature.

We are interested in extending this model in several ways. For instance, we would like to model the effects of HIV-1 proteins on bystander cell apoptosis. As mentioned in the introduction, HIV-1 appears to primarily kill uninfected bystander T cells [15]. Various mechanisms have been reported for the destruction of the bystander cells. Along with Fas-induced apoptosis, other possible mechanisms for bystander cell death are reviewed in [2,38,41]. Upon being exocytosed by an infected cell, several of the proteins encoded in HIV-1 can exhibit destructive qualities when interacting with neighboring bystander cells – either on the surface or through endocytosis.

There are a few HIV-1 proteins we have ignored in this model, because they affect T cells in ways not within the scope of our current efforts. For example, soluble and membrane-bound Env can bind to the CD4 receptor of bystander cells. In [11] and [5], the authors have shown that ligation of the CD4 receptor by Env, is sufficient to increase apoptosis in bystander cells. The reasons for the increased apoptotic rates following Env-CD4 binding can be attributed to Bcl-2 down-regulation [20], increased Caspase 8 activation [1], and upregulation of Fas [33], FasL and Bax [41].

Extracellular Tat can enter bystander cells through endocytosis, which leads to pro-apoptotic activity. The addition of Tat to a culture of uninfected cells has been shown to increase apoptosis [29]. Endocytosed Tat can upregulate levels of Caspase 8 [4] and increase expression of the Fas ligand [41], interfering in the same manner as in the infected cells. Also, extracellular Vpr can disrupt the mitochondrial membrane, leading to increased translocation of Cytochrome c* [41].

Finally, we would like to note that the experimental information on the latent HIV-1-infected CD4+ T cells is scarce, due to the fact that these cells are found in such small numbers *in vivo*. Therefore, our model relies heavily on applying the knowledge of activated HIV-1-infected CD4+ T cells. We look forward to new experimental results about these enigmatic cells, which we will use to refine the model.

Acknowledgements. We gratefully acknowledge support in part from a NSF GK-12 Ph.D. fellowship, support from NSF Grant CCF-0523572, support from LA BoR RSC grant LEQSF (2004-07)-RD-A-23, support from INBRE Program of the NCRR (a division of NIH), support from CNCSIS grant RP-13, support from CNMP grant 11-56 /2007, support from Spanish Ministry of Science and Education (MEC) under project TIN2006-15595, and support from the Comunidad de Madrid (grant No. CCG07-UPM/TIC-0386 to the LIA research group).

Bibliography

[1] Algeciras-Schimnich, A. et al.: CCR5 Mediates Fas- and Caspase 8 Dependent Apoptosis of Both Uninfected and HIV Infected Primary Human CD4 T cells. AIDS. 16, 1467-1478 (2002)

[2] Alimonti, J. et al.: Mechanisms of CD4 T Lymphocyte cell death in human immunodeficiency virus infection and AIDS. J. Gen. Vir. 84, 1649-1661 (2003)

[3] Ashkenazi, A., Dixit, V.: Death receptors: signaling and modulation. Science. 281, 1305-1308 (1998)

[4] Bartz, S., Emerman, M.: Human immunodeficiency virus type 1 Tat induces apoptosis and increases sensitivity to apoptotic signals by up-regulating FLICE/Caspase 8. J. Vir. 73, 1956 1963 (1999)

[5] Biard-Piechaczyk M., et al.: Caspase-dependent apoptosis of cells expressing the chemokine receptor CXCR4 is induced by cell membrane-associated human immunodeficiency virus type 1 envelope glycoprotein (gp120). Vir. 268, 329-344 (2000)

[6] Briggs, J. et. al.: The stoichiometry of Gag protein in HIV-1. Nature Struct. Mol. Bio. 11, 672-675 (2004)

[7] Blankson, J. N. et al.: Biphasic decay of latently infected CD4+ T cells in acute HIV-1 infection. J. Infect. Dis. 182, 16361642 (2000).

[8] Chan, D., Kim, P.: HIV entry and its inhibition. Cell. 93, 681-684 (1998)

[9] Chun, T. W. et al.: In vivo fate of HIV-1-infected T cells: quantitative analysis of the transition to stable latency. Nature Med. 1, 12841290 (1995)

[10] Chun, T. W. et al.: Quantification of latent tissue reservoirs and total body viral load in HIV-1 infection. Nature. 387, 183188 (1997)

[11] Cicala C., et al.: HIV-1 envelope induces activation of caspase-3 and cleavage of focal adhesion kinase in primary human CD4(+) T cells. Proc. Natl. Acad. Sci. U.S.A. 97, 1178-1183 (2000)

[12] Conti, L. et al.: The HIV-1 vpr protein acts as a negative regulator of apoptosis in a human lymphoblastoid T cell line possible implications for the pathogenesis of aids. J. Exp. Med. 187, 403-413 (1998)

[13] Crise, B, et al.: CD4 is retained in the ER by the human immunodeficiency virus type 1 glycoprotein precursor. J. Vir. 64, 5585-5593 (1990)

[14] Farnet, C., Haseltine, W.: Determination of viral proteins present in the human immunodeficiency virus type 1 preintegration complex. J. Vir. 65, 1910-1915 (1991)

[15] Finkel, T.: Apoptosis occurs predominantly in bystander cells and not in productively infected cells of HIV- and SIV-infected lymph nodes. Nature Med. 1, 129-134 (1995)

[16] Finzi, D. et al.: Latent infection of CD4+ T cells provides a mechanism for lifelong persistence of HIV-1, even in patients on effective combination therapy. Nature Med. 5, 512517 (1999).

[17] Gillespie, D.T.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. J. Comp. Phy. 22, 403-434 (1976)

[18] Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. The J. Phy. Chem. 81, 2340-2361 (1977)
[19] Han, T., et al.: Experimental approaches to the study of HIV-1 latency. Microbio. 5, 95-106 (2007)
[20] Hashimoto, F., et al.: Modulation of Bcl-2 Protein by CD4 Cross-Linking a possible mechanism for lympphocyte apoptosis in human immunodeficiency virus infection. Blood. 90, 745-753 (1997)
[21] Hua, F., et al.: Effects of Bcl-2 levels on FAS signaling-induced caspase-3 activation: molecular genetic tests of computational model predictions. J. of Immun., 175, 985-995 (2005) and correction 175, 6235-6237 (2005)
[22] Igney, F., Krammer, P.: Death and anti-death: tumour resistance to apoptosis. Nature Rev. Can. 2, 277-288 (2002)
[23] Jack et al. Discrete nondeterministic modeling of the Fas pathway. IJFCS (14pp) (2008) [accepted]
[24] Jacotet E., et al.: The HIV-1 viral protein R induces apoptosis via a direct effect on the mitochondrial permeability transition pore. J. Exp. Med. 191, 33-45 (2000)
[25] Kim, S., et al.: Temporal aspects of DNA and RNA synthesis during human immunodeficiency virus infection: evidence for differential gene expression. J. Vir. 63, 3708-3713 (1989)
[26] Kinoshita, S., et al.: The T cell activation factor NF-ATc positively regulates HIV-1 replication and gene expression in T cells. Immun. 6, 235-244 (1997)
[27] Lassen, K., et al.: Nuclear retention of multiply spliced HIV-1 RNA in resting CD4+ T cells. PLoS Pathogens. 2, 650-661 (2006)
[28] Liu, X., et al.: DFF, a heterodimeric protein that functions downstream of Caspase-3 to trigger DNA fragmentation during apoptosis. Cell. 89, 175-184 (1997)
[29] McCloskey, T., et al.: Dual role of HIV Tat in regulation of apoptosis in T cells. J. Immun. 158, 1014-1019 (1997)
[30] Nabel, G., Baltimore, D.: An inducible transcription factor activates expression of human immunodeficiency virus in T cells. Nature. 344, 711-713 (1987)
[31] Napolitano, G., et al.: CDK9 has the intrinsic property to shuttle between nucleus and cytoplasm, and enhanced expression of CyclinT1 promotes its nuclear localization. J Cell. Phys. 192, 209-215 (2002)
[32] Nie, Z. et al.: HIV-1 protease processes procaspase 8 to cause mitochondrial release of cytochrome c, caspase cleavage and nuclear fragmentation. Cell Death Diff. 9, 1172-1184 (2002)
[33] Oyaizu, N., et al.: Cross-linking of CD4 molecules upregulates Fas antigen expression in lymphocytes by inducing interferon-gamma and tumor necrosis factor-alpha secretion. Blood. 84, 2622-2631 (1994)
[34] Perelson, A.S. et al.: Decay characteristics of HIV-1-infected compartments during combination therapy. Nature. 387, 188-191 (1997)
[35] Pierson, T. C. et al.: Molecular characterization of preintegration latency in HIV-1 infection. J. Virol. 76, 85188531 (2002)
[36] Rieux-Laucat, F., et al.: Autoimmune lymphoproliferative syndromes: genetic defects of apoptosis pathways. Cell Death Diff. 10, 124-133 (2003)

[37] Roland-Jones, S.L., Whittle, H.C.: Out of Africa: what can we learn from HIV-2 about protective immunity to HIV-1. Nature Imm. 9, 329-331 (2007)
[38] Ross, T. Using death to one's advantage: HIV modulation and apoptosis. Leuk. 15, 332-341 (2001)
[39] Salghetti, S, et al.: Human immunodeficiency virus type 1 Nef and p56Ick protein-tyrosine kinase interact with a common element in CD4 cytoplasmic tail. Proc. Natl. Acad. Sci. U.S.A. 92, 349-353 (1995)
[40] Scaffidi, C., et al. Two CD95 (APO-1/Fas) signaling pathways. EMBO J. 17, 1675-1687 (1998)
[41] Selliah, N., Finkel, T.: Biochemical mechanisms of HIV induced T cell apoptosis. Cell Death Diff. 8, 127-136 (2001)
[42] Siliciano, J. D. et al.: Long-term follow-up studies confirm the stability of the latent reservoir for HIV-1 in resting CD4+ T cells. Nature Med. 9, 727728 (2003)
[43] Strack, et al.: Apoptosis mediated by HIV protease is preceded by cleavage of Bcl-2. Proc. Natl. Acad. Sci., U.S.A. 93, 9571-9576 (1996)
[44] Strain, M. C. et al.: Heterogeneous clearance rates of long-lived lymphocytes infected with HIV: intrinsic stability predicts lifelong persistence. Proc. Natl Acad. Sci. U.S.A. 100, 48194824 (2003).
[45] Wildum, S. et al.: Contribution of Vpu, Env, and Nef to CD4 down-modulation and resistance of human immunodeficiency virus type 1-infected T cells to superinfection. J. Vir. 80, 8047-8059 (2006)
[46] Willey, R. et al.: Human immunodeficiency virus type 1 vpu protein induces rapid degradation of CD4 J. Vir. 66, 7193-7200 (1992)
[47] World Health Organization 2007 AIDS Epidemic Update. http://www.who.int/hiv/en/
[48] Wyatt, R., Sodroski, J.: The HIV-1 envelope glycoproteins: fusogens, antigens, and immunogens. Science. 280, 5371 1884-1888 (1998)
[49] Yang, Y. et al.: HIV Tat binds Egr proteins and enhances Egr-dependent transactivation of the Fas ligand promoter. J. Bio. Chem. 277, 19482-19487 (2002)
[50] Zack, J. A. et al.: HIV-1 entry into quiescent primary lymphocytes: molecular analysis reveals a labile, latent viral structure. Cell. 61, 213222 (1990)
[51] Zack, J. A., et al.: Incompletely reverse-transcribed human immunodeficiency virus type I genomes function as intermediates in the retroviral life cycle. J. Vir. 66, 17171725 (1992)
[52] Zauli, G., et al. Human immunodeficiency virus type 1 Nef protein sensitizes CD4+ T lymphoid cells to apoptosis via functional upregulation of the CD95/CD95 ligand pathway. Blood. 93, 1000-1010 (1999)
[53] Zhou, Y., et al.: Kinetics of human immunodeficiency virus type 1 decay following entry into resting CD4+ T cells. J. Vir. 79, 21992210 (2005)

label	rule	rate
r_1 :	$FASL[\ FAS\ ]_s \rightarrow [\ FASC\ ]_s$	k_{1f}
r_2 :	$[\ FASC\ ]_s \rightarrow FASL[\ FASC\ ]_s$	k_{1r}
r_3 :	$FASC[\ FADD\ ]_c \rightarrow FASC : FADD[\ \ ]_c$	k_{2f}
r_4 :	$FASC : FADD[\ \ ]_c \rightarrow FASC[\ FADD\ ]_c$	k_{2r}
r_5 :	$FASC : FADD[\ FADD\ ]_c \rightarrow FASC : FADD_2[\ \ ]_c$	k_{2f}
r_6 :	$FASC : FADD_2[\ \ ]_c \rightarrow FASC : FADD[\ FADD\ ]_c$	k_{2r}
r_7 :	$FASC : FADD_2[\ FADD\ ]_c \rightarrow FASC : FADD_3[\ \ ]_c$	k_{2f}
r_8 :	$FASC : FADD_3[\ \ ]_c \rightarrow FASC : FADD_2[\ FADD\ ]_c$	k_{2r}
r_9 :	$FASC : FADD_2 : CASP8[\ FADD\ ]_c \rightarrow FASC : FADD_3 : CASP8[\ \ ]_c$	k_{2f}
r_{10} :	$FASC : FADD_3 : CASP8[\ \ ]_c \rightarrow FASC : FADD_2 : CASP8[\ FADD\ ]_c$	k_{2r}
r_{11} :	$FASC : FADD_2 : FLIP[\ FADD\ ]_c \rightarrow FASC : FADD_3 : FLIP[\ \ ]_c$	k_{2f}
r_{12} :	$FASC : FADD_3 : FLIP[\ \ ]_c \rightarrow FASC : FADD_2 : FLIP[\ FADD\ ]_c$	k_{2r}
r_{13} :	$FASC : FADD_2 : CASP8_2[\ FADD\ ]_c \rightarrow FASC : FADD_3 : CASP8_2[\ \ ]_c$	k_{2f}
r_{14} :	$FASC : FADD_3 : CASP8_2[\ \ ]_c \rightarrow FASC : FADD_2 : CASP8_2[\ FADD\ ]_c$	k_{2r}
r_{15} :	$FASC : FADD_2 : CASP8 : FLIP[\ FADD\ ]_c \rightarrow FASC : FADD_3 : CASP8 : FLIP[\ \ ]_c$	k_{2f}
r_{16} :	$FASC : FADD_3 : CASP8 : FLIP[\ \ ]_c \rightarrow FASC : FADD_2 : CASP8 : FLIP[\ FADD\ ]_c$	k_{2r}
r_{17} :	$FASC : FADD_2 : FLIP_2[\ FADD\ ]_c \rightarrow FASC : FADD_3 : FLIP_2[\ \ ]_c$	k_{2f}
r_{18} :	$FASC : FADD_3 : FLIP_2[\ \ ]_c \rightarrow FASC : FADD_2 : FLIP_2[\ FADD\ ]_c$	k_{2r}
r_{19} :	$FASC : FADD : CASP8[\ FADD\ ]_c \rightarrow FASC : FADD_2 : CASP8[\ \ ]_c$	k_{2f}
r_{20} :	$FASC : FADD_2 : CASP8[\ \ ]_c \rightarrow FASC : FADD : CASP8[\ FADD\ ]_c$	k_{2r}
r_{21} :	$FASC : FADD : FLIP[\ FADD\ ]_c \rightarrow FASC : FADD_2 : FLIP[\ \ ]_c$	k_{2f}
r_{22} :	$FASC : FADD_2 : FLIP[\ \ ]_c \rightarrow FASC : FADD : FLIP[\ FADD\ ]_c$	k_{2r}
r_{23} :	$FASC : FADD_3[\ CASP8\ ]_c \rightarrow FASC : FADD_3 : CASP8[\ \ ]_c$	k_{2f}
r_{24} :	$FASC : FADD_3 : CASP8[\ \ ]_c \rightarrow FASC : FADD_3[\ CASP8\ ]_c$	k_{2r}
r_{25} :	$FASC : FADD_3[\ FLIP\ ]_c \rightarrow FASC : FADD_3 : FLIP[\ \ ]_c$	k_{3f}
r_{26} :	$FASC : FADD_3 : FLIP[\ \ ]_c \rightarrow FASC : FADD_3[\ FLIP\ ]_c$	k_{3r}
r_{27} :	$FASC : FADD_3 : CASP8[\ CASP8\ ]_c \rightarrow FASC : FADD_3 : CASP8_2[\ \ ]_c$	k_{3f}
r_{28} :	$FASC : FADD_3 : CASP8_2[\ \ ]_c \rightarrow FASC : FADD_3 : CASP8[\ CASP8\ ]_c$	k_{3r}
r_{29} :	$FASC : FADD_3 : CASP8[\ FLIP\ ]_c \rightarrow FASC : FADD_3 : CASP8 : FLIP[\ \ ]_c$	k_{3f}
r_{30} :	$FASC : FADD_3 : CASP8 : FLIP[\ \ ]_c \rightarrow FASC : FADD_3 : CASP8[\ FLIP\ ]_c$	k_{3r}
r_{31} :	$FASC : FADD_3 : FLIP[\ CASP8\ ]_c \rightarrow FASC : FADD_3 : CASP8 : FLIP[\ \ ]_c$	k_{3f}
r_{32} :	$FASC : FADD_3 : CASP8 : FLIP[\ \ ]_c \rightarrow FASC : FADD_3 : FLIP[\ CASP8\ ]_c$	k_{3r}
r_{33} :	$FASC : FADD_3 : FLIP[\ FLIP\ ]_c \rightarrow FASC : FADD_3 : FLIP_2[\ \ ]_c$	k_{3f}
r_{34} :	$FASC : FADD_3 : FLIP_2[\ \ ]_c \rightarrow FASC : FADD_3 : FLIP[\ FLIP\ ]_c$	k_{3r}
r_{35} :	$FASC : FADD_3 : CASP8_2[\ CASP8\ ]_c \rightarrow FASC : FADD_3 : CASP8_3[\ \ ]_c$	k_{3f}
r_{36} :	$FASC : FADD_3 : CASP8_3[\ \ ]_c \rightarrow FASC : FADD_3 : CASP8_2[\ CASP8\ ]_c$	k_{3r}
r_{37} :	$FASC : FADD_3 : CASP8_2[\ FLIP\ ]_c \rightarrow FASC : FADD_3 : CASP8_2 : FLIP[\ \ ]_c$	k_{3f}
r_{38} :	$FASC : FADD_3 : CASP8_2 : FLIP[\ \ ]_c \rightarrow FASC : FADD_3 : CASP8_2[\ FLIP\ ]_c$	k_{3r}
r_{39} :	$FASC : FADD_3 : CASP8 : FLIP[\ CASP8\ ]_c \rightarrow FASC : FADD_3 : CASP8_2 : FLIP[\ \ ]_c$	k_{3f}
r_{40} :	$FASC : FADD_3 : CASP8_2 : FLIP[\ \ ]_c \rightarrow FASC : FADD_3 : CASP8 : FLIP[\ CASP8\ ]_c$	k_{3r}
r_{41} :	$FASC : FADD_3 : CASP8 : FLIP[\ FLIP\ ]_c \rightarrow FASC : FADD_3 : CASP8 : FLIP_2[\ \ ]_c$	k_{3f}
r_{42} :	$FASC : FADD_3 : CASP8 : FLIP_2[\ \ ]_c \rightarrow FASC : FADD_3 : CASP8 : FLIP[\ FLIP\ ]_c$	k_{3r}
r_{43} :	$FASC : FADD_3 : FLIP_2[\ CASP8\ ]_c \rightarrow FASC : FADD_3 : CASP8 : FLIP_2[\ \ ]_c$	k_{3f}
r_{44} :	$FASC : FADD_3 : CASP8 : FLIP_2[\ \ ]_c \rightarrow FASC : FADD_3 : FLIP_2[\ CASP8\ ]_c$	k_{3r}

label	rule	rate
r_{45}	$FASC : FADD_3 : FLIP_2[\ FLIP\ ]_c \rightarrow FASC : FADD_3 : FLIP_3[\ \ ]_c$	k_{3f}
r_{46}	$FASC : FADD_3 : FLIP_3[\ \ ]_c \rightarrow FASC : FADD_3 : FLIP_2[\ FLIP\ ]_c$	k_{3r}
r_{47}	$FASC : FADD_2[\ CASP8\ ]_c \rightarrow FASC : FADD_2 : CASP8[\ \ ]_c$	k_{3f}
r_{48}	$FASC : FADD_2 : CASP8[\ \ ]_c \rightarrow FASC : FADD_2[\ CASP8\ ]_c$	k_{3r}
r_{49}	$FASC : FADD_2[\ FLIP\ ]_c \rightarrow FASC : FADD_2 : FLIP[\ \ ]_c$	k_{3f}
r_{50}	$FASC : FADD_2 : FLIP[\ \ ]_c \rightarrow FASC : FADD_2[\ FLIP\ ]_c$	k_{3r}
r_{51}	$FASC : FADD_2 : CASP8[\ CASP8\ ]_c \rightarrow FASC : FADD_2 : CASP8_2[\ \ ]_c$	k_{3f}
r_{52}	$FASC : FADD_2 : CASP8_2[\ \ ]_c \rightarrow FASC : FADD_2 : CASP8[\ CASP8\ ]_c$	k_{3r}
r_{53}	$FASC : FADD_2 : CASP8[\ FLIP\ ]_c \rightarrow FASC : FADD_2 : CASP8 : FLIP[\ \ ]_c$	k_{3f}
r_{54}	$FASC : FADD_2 : CASP8 : FLIP[\ \ ]_c \rightarrow FASC : FADD_2 : CASP8[\ FLIP\ ]_c$	k_{3r}
r_{55}	$FASC : FADD_2 : FLIP[\ CASP8\ ]_c \rightarrow FASC : FADD_2 : CASP8 : FLIP[\ \ ]_c$	k_{3f}
r_{56}	$FASC : FADD_2 : CASP8 : FLIP[\ \ ]_c \rightarrow FASC : FADD_2 : FLIP[\ CASP8\ ]_c$	k_{3r}
r_{57}	$FASC : FADD_2 : FLIP[\ FLIP\ ]_c \rightarrow FASC : FADD_2 : FLIP_2[\ \ ]_c$	k_{3f}
r_{58}	$FASC : FADD_2 : FLIP_2[\ \ ]_c \rightarrow FASC : FADD_2 : FLIP[\ FLIP\ ]_c$	k_{3r}
r_{59}	$FASC : FADD[\ CASP8\ ]_c \rightarrow FASC : FADD : CASP8[\ \ ]_c$	k_{3f}
r_{60}	$FASC : FADD : CASP8[\ \ ]_c \rightarrow FASC : FADD[\ CASP8\ ]_c$	k_{3r}
r_{61}	$FASC : FADD[\ FLIP\ ]_c \rightarrow FASC : FADD : FLIP[\ \ ]_c$	k_{3f}
r_{62}	$FASC : FADD : FLIP[\ \ ]_c \rightarrow FASC : FADD[\ FLIP\ ]_c$	k_{3r}
r_{63}	$FASC : FADD_2 : CASP8_2[\ \ ]_c \rightarrow FASC : FADD_2[\ CASP8_2^{P41}\ ]_c$	k_4
r_{64}	$FASC : FADD_3 : CASP8_3[\ \ ]_c \rightarrow FASC : FADD_3 : CASP8[\ CASP8_2^{P41}\ ]_c$	k_4
r_{65}	$FASC : FADD_3 : CASP8_2 : FLIP[\ \ ]_c \rightarrow FASC : FADD_3 : FLIP[\ CASP8_2^{P41}\ ]_c$	k_4
r_{66}	$FASC : FADD_3 : CASP8_2[\ \ ]_c \rightarrow FASC : FADD_3[\ CASP8_2^{P41}\ ]_c$	k_4
r_{67}	$[\ CASP8_2^{P41}\ ]_c \rightarrow [\ CASP8_2^*\ ]_c$	k_5
r_{68}	$[\ CASP8_2^*, CASP3\ ]_c \rightarrow [\ CASP8_2^* : CASP3\ ]_c$	k_{6f}
r_{69}	$[\ CASP8_2^* : CASP3\ ]_c \rightarrow [\ CASP8_2^*, CASP3\ ]_c$	k_{6r}
r_{70}	$[\ CASP8_2^* : CASP3\ ]_c \rightarrow [\ CASP8_2^*, CASP3^*\ ]_c$	k_7
r_{71}	$[\ CASP3^*, XIAP\ ]_c \rightarrow [\ CASP3^* : XIAP\ ]_c$	k_{19f}
r_{72}	$[\ CASP3^* : XIAP\ ]_c \rightarrow [\ CASP3^*, XIAP\ ]_c$	k_{19r}
r_{73}	$[\ CASP8_2^*, Bid\ ]_c \rightarrow [\ CASP8_2^* : Bid\ ]_c$	k_{8f}
r_{74}	$[\ CASP8_2^* : Bid\ ]_c \rightarrow [\ CASP8_2^*, Bid\ ]_c$	k_{8r}
r_{75}	$[\ CASP8_2^* : Bid\ ]_c \rightarrow [\ CASP8_2^*, tBid\ ]_c$	k_7
r_{76}	$[\ tBid, Bax\ ]_c \rightarrow [\ tBid : Bax\ ]_c$	k_{9f}
r_{77}	$[\ tBid : Bax\ ]_c \rightarrow [\ tBid, Bax\ ]_c$	k_{9r}
r_{78}	$[\ tBid : Bax, Bax\ ]_c \rightarrow [\ tBid : Bax_2\ ]_c$	k_{9f}
r_{79}	$[\ tBid : Bax_2\ ]_c \rightarrow [\ tBid : Bax, Bax\ ]_c$	k_{9r}
r_{80}	$tBid : Bax_2[\ Smac\ ]_m \rightarrow tBid : Bax_2, Smac^*[\ \ ]_m$	k_{10}
r_{81}	$tBid : Bax_2[\ Cyto.c\ ]_m \rightarrow tBid : Bax_2, Cyto.c^*[\ \ ]_m$	k_{10}
r_{82}	$[\ Smac^*, XIAP\ ]_c \rightarrow [\ Smac^* : XIAP\ ]_c$	k_{11f}
r_{83}	$[\ Smac^* : XIAP\ ]_c \rightarrow [\ Smac^*, XIAP\ ]_c$	k_{11r}
r_{84}	$[\ Cyto.c^*, Apaf, ATP\ ]_c \rightarrow [\ Cyto.c^* : Apaf : ATP\ ]_c$	k_{12f}
r_{85}	$[\ Cyto.c^* : Apaf : ATP\ ]_c \rightarrow [\ Cyto.c^*, Apaf, ATP\ ]_c$	k_{12r}
r_{86}	$[\ Cyto.c^* : Apaf : ATP, CASP9\ ]_c \rightarrow [\ Cyto.c^* : Apaf : ATP : CASP9\ ]_c$	k_{13f}
r_{87}	$[\ Cyto.c^* : Apaf : ATP : CASP9\ ]_c \rightarrow [\ Cyto.c^* : Apaf : ATP, CASP9\ ]_c$	k_{13r}
r_{88}	$[\ Cyto.c^* : Apaf : ATP : CASP9, CASP9\ ]_c \rightarrow [\ Cyto.c^* : Apaf : ATP : CASP9_2\ ]_c$	k_{14f}

label	rule	rate
r_{89} :	$[\ Cyto.c^* : Apaf : ATP : CASP9_2\ ]_c \rightarrow [\ Cyto.c^* : Apaf : ATP : CASP9, CASP9\ ]_c$	k_{14r}
r_{90} :	$[\ Cyto.c^* : Apaf : ATP : CASP9_2\ ]_c \rightarrow [\ Cyto.c^* : Apaf : ATP : CASP9, CASP9^*\ ]_c$	k_{15}
r_{91} :	$[\ CASP9^*, CASP3\ ]_c \rightarrow [\ CASP9^* : CASP3\ ]_c$	k_{16f}
r_{92} :	$[\ CASP9^* : CASP3\ ]_c \rightarrow [\ CASP9^*, CASP3\ ]_c$	k_{16r}
r_{93} :	$[\ CASP9^* : CASP3\ ]_c \rightarrow [\ CASP9^*, CASP3^*\ ]_c$	k_{17}
r_{92} :	$[\ CASP9, XIAP\ ]_c \rightarrow [\ CASP9 : XIAP\ ]_c$	k_{18f}
r_{93} :	$[\ CASP9 : XIAP\ ]_c \rightarrow [\ CASP9, XIAP\ ]_c$	k_{18r}
r_{96} :	$Bax[\ Bcl2\ ]_m \rightarrow [\ Bcl2 : Bax\ ]_m$	k_{20f}
r_{97} :	$[\ Bcl2 : Bax\ ]_m \rightarrow Bax[\ Bcl2\ ]_m$	k_{20r}
r_{98} :	$tBid[\ Bcl2\ ]_m \rightarrow [\ Bcl2 : tBid\ ]_m$	k_{20f}
r_{99} :	$[\ Bcl2 : tBid\ ]_m \rightarrow tBid[\ Bcl2\ ]_m$	k_{20r}
r_{100} :	$[\ HIV_{RNA}, RT\ ]_c \rightarrow [\ HIV_{cDNA}, RT\ ]_m$	k_{21}
r_{101} :	$HIV_{cDNA}[\ \ ]_n \rightarrow [\ HIV_{cDNA}\ ]_n$	k_{22}
r_{102} :	$[\ HIV_{cDNA}\ ]_n \rightarrow [\ HIV_{LTR}\ ]_n$	k_{22}
r_{103} :	$NFAT[\ \ ]_n \rightarrow [\ NFAT\ ]_n$	k_{23}
r_{104} :	$CDK9[\ \ ]_n \rightarrow [\ CDK9\ ]_n$	k_{24}
r_{105} :	$[\ CyclinT1, CDK9\ ]_n \rightarrow [\ PTEFb\ ]_n$	k_{25}
r_{106} :	$[\ NFAT, HIV_{LTR}\ ]_n \rightarrow [\ HIV_{LTR} : NFAT\ ]_n$	k_{26}
r_{107} :	$[\ HIV_{LTR} : NFAT, Tat\ ]_n \rightarrow [\ HIV_{LTR} : NFAT : Tat\ ]_n$	k_{27}
r_{108} :	$[\ HIV_{LTR} : NFAT : Tat, PTEFb\ ]_n \rightarrow [\ HIV_{LTR} : NFAT : Tat : PTEFb\ ]_n$	k_{28}
r_{109} :	$[\ HIV_{LTR}\ ]_n \rightarrow [\ HIV_{LTR}, mRNA_{Tat}\ ]_n$	k_{29}
r_{110} :	$[\ HIV_{LTR}\ ]_n \rightarrow [\ HIV_{LTR}, mRNA_{Vpr}\ ]_n$	k_{29}
r_{111} :	$[\ HIV_{LTR}\ ]_n \rightarrow [\ HIV_{LTR}, mRNA_{HIV_{pr}}\ ]_n$	k_{29}
r_{112} :	$[\ HIV_{LTR}\ ]_n \rightarrow [\ HIV_{LTR}, mRNA_{Nef}\ ]_n$	k_{29}
r_{113} :	$[\ HIV_{LTR} : NFAT\ ]_n \rightarrow [\ HIV_{LTR} : NFAT, mRNA_{Tat}\ ]_n$	k_{30}
r_{114} :	$[\ HIV_{LTR} : NFAT\ ]_n \rightarrow [\ HIV_{LTR} : NFAT, mRNA_{Vpr}\ ]_n$	k_{30}
r_{115} :	$[\ HIV_{LTR} : NFAT\ ]_n \rightarrow [\ HIV_{LTR} : NFAT, mRNA_{HIV_{pr}}\ ]_n$	k_{30}
r_{116} :	$[\ HIV_{LTR} : NFAT\ ]_n \rightarrow [\ HIV_{LTR} : NFAT, mRNA_{Nef}\ ]_n$	k_{30}
r_{117} :	$[\ HIV_{LTR} : NFAT : Tat\ ]_n \rightarrow [\ HIV_{LTR} : NFAT : Tat, mRNA_{Tat}\ ]_n$	k_{31}
r_{118} :	$[\ HIV_{LTR} : NFAT : Tat\ ]_n \rightarrow [\ HIV_{LTR} : NFAT : Tat, mRNA_{Vpr}\ ]_n$	k_{31}
r_{119} :	$[\ HIV_{LTR} : NFAT : Tat\ ]_n \rightarrow [\ HIV_{LTR} : NFAT : Tat, mRNA_{HIV_{pr}}\ ]_n$	k_{31}
r_{120} :	$[\ HIV_{LTR} : NFAT : Tat\ ]_n \rightarrow [\ HIV_{LTR} : NFAT : Tat, mRNA_{Nef}\ ]_n$	k_{31}
r_{121} :	$[\ HIV_{LTR} : NFAT : Tat : PTEFb\ ]_n \rightarrow [\ HIV_{LTR} : NFAT : Tat : PTEFb, mRNA_{Tat}\ ]_n$	k_{32}
r_{122} :	$[\ HIV_{LTR} : NFAT : Tat : PTEFb\ ]_n \rightarrow [\ HIV_{LTR} : NFAT : Tat : PTEFb, mRNA_{Vpr}\ ]_n$	k_{32}
r_{123} :	$[\ HIV_{LTR} : NFAT : Tat : PTEFb\ ]_n \rightarrow [\ HIV_{LTR} : NFAT : Tat : PTEFb, mRNA_{HIV_{pr}}\ ]_n$	k_{32}
r_{124} :	$[\ HIV_{LTR} : NFAT : Tat : PTEFb\ ]_n \rightarrow [\ HIV_{LTR} : NFAT : Tat : PTEFb, mRNA_{Nef}\ ]_n$	k_{32}
r_{125} :	$[\ mRNA_{Tat}\ ]_n \rightarrow mRNA_{Tat}[\ \ ]_n$	k_{33}
r_{126} :	$[\ mRNA_{Nef}\ ]_n \rightarrow mRNA_{Nef}[\ \ ]_n$	k_{33}
r_{127} :	$[\ mRNA_{Vpr}\ ]_n \rightarrow mRNA_{Vpr}[\ \ ]_n$	k_{33}
r_{128} :	$[\ mRNA_{HIV_{pr}}\ ]_n \rightarrow mRNA_{HIV_{pr}}[\ \ ]_n$	k_{33}
r_{129} :	$[\ mRNA_{Tat}\ ]_c \rightarrow [\ mRNA_{Tat}, Tat\ ]_n$	k_{34}
r_{130} :	$[\ mRNA_{Nef}\ ]_c \rightarrow [\ mRNA_{Nef}, Nef\ ]_n$	k_{34}
r_{131} :	$[\ mRNA_{Vpr}\ ]_c \rightarrow [\ mRNA_{Vpr}, Vpr\ ]_n$	k_{34}
r_{132} :	$[\ mRNA_{HIV_{pr}}\ ]_c \rightarrow [\ mRNA_{HIV_{pr}}, HIV_{pr}\ ]_n$	k_{34}
r_{133} :	$[\ mRNA_{Tat}\ ]_c \rightarrow [\ \ ]_c$	k_{35}

label	rule	rate
r_{134}	$[\ mRNA_{Nef}\ ]_c \rightarrow [\ \ ]_c$	k_{35}
r_{135}	$[\ mRNA_{Vpr}\ ]_c \rightarrow [\ \ ]_c$	k_{35}
r_{136}	$[\ mRNA_{HIV_{pr}}\ ]_c \rightarrow [\ \ ]_c$	k_{35}
r_{137}	$Tat[\ \ ]_n \rightarrow [\ Tat\ ]_n$	k_{36f}
r_{138}	$[\ Tat\ ]_n \rightarrow [\ Tat\ ]_n Casp8$	k_{37}
r_{139}	$[\ Tat\ ]_n \rightarrow [\ Tat\ ]_n Bcl2$	k_{38}
r_{140}	$[\ Tat\ ]_n \rightarrow FASL[\ Tat\ ]_n$	k_{39}
r_{141}	$[\ Tat, Bcl2\ ]_c \rightarrow [\ Tat\ ]_c$	k_{40}
r_{142}	$Vpr[\ \ ]_m \rightarrow Vpr[\ Bcl2\ ]_m$	k_{41}
r_{143}	$[\ Vpr, Bax\ ]_c \rightarrow [\ Vpr\ ]_c$	k_{42}
r_{144}	$[\ Vpr, Bcl2\ ]_m \rightarrow [\ Vpr : Bcl2\ ]_m$	k_{43}
r_{145}	$[\ Vpr : Bcl2\ ]_m \rightarrow [\ Vpr, Bcl2\ ]_m$	k_{44}
r_{146}	$[\ Vpr, PTPC\ ]_m \rightarrow [\ Vpr : PTPC\ ]_m$	k_{45}
r_{147}	$[\ Vpr : PTPC, Cyto.c\ ]_m \rightarrow Cyto.c^*[\ Vpr : PTPC\ ]_m$	k_{46}
r_{148}	$[\ HIVProtease, Casp8\ ]_c \rightarrow [\ HIVProtease, Casp8*\ ]_m$	k_{47}
r_{149}	$HIVProtease[\ Bcl2\ ]_m \rightarrow HIVProtease[\ \ ]_m$	k_{48}
r_{150}	$[\ Nef\ ]_n \rightarrow [\ Nef, FAS\ ]_m$	k_{49}
r_{151}	$[\ Nef\ ]_n \rightarrow [\ Nef, FASL\ ]_m$	k_{50}
r_{152}	$[\ FASL\ ]_s \rightarrow FASL[\ \ ]_s$	k_{51}
r_{153}	$[\ Tat\ ]_n \rightarrow Tat[\ \ ]_n$	k_{36r}

The following tables give the deterministic kinetic rates (reaction rates) used in the description of the reactions;

$k_{1f} = 9.09E-05 nM^{-1}s^{-1}$	$k_{11r} = 2.21E-03\ s^{-1}$
$k_{1r} = 1.00E-04 s^{-1}$	$k_{12f} = 2.78E-07 nM^{-1}s^{-1}nM^{-1}$
$k_{2f} = 5.00E-04 nM^{-1}s^{-1}$	$k_{12r} = 5.70E-03\ s^{-1}$
$k_{2r} = 0.2 s^{-1}$	$k_{13f} = 2.84E-04 nM^{-1}s^{-1}$
$k_{3f} = 3.50E-03 nM^{-1}s^{-1}$	$k_{13r} = 0.07493 s^{-1}$
$k_{3r} = 0.018 s^{-1}$	$k_{14f} = 4.41E-04 nM^{-1}s^{-1}$
$k_4 = 0.3 s^{-1}$	$k_{14r} = 0.1 s^{-1}$
$k_5 = 0.1 s^{-1}$	$k_{15} = 0.7 s^{-1}$
$k_{6f} = 1.00E-05 nM^{-1}s^{-1}$	$k_{16f} = 1.96E-05 nM^{-1}s^{-1}$
$k_{6r} = 0.06 s^{-1}$	$k_{16r} = 0.05707 s^{-1}$
$k_7 = 0.1 s^{-1}$	$k_{17} = 4.8 s^{-1}$
$k_{8f} = 5.00E-03 nM^{-1}s^{-1}$	$k_{18f} = 1.06E-04 nM^{-1}s^{-1}$
$k_{8r} = 0.005\ s^{-1}$	$k_{18r} = 1.00E-03 s^{-1}$
$k_{9f} = 2.00E-04 nM^{-1}s^{-1}$	$k_{19f} = 2.47E-03 nM^{-1}s^{-1}$
$k_{9r} = 0.02 s^{-1}$	$k_{19r} = 2.40E-03 s^{-1}$
$k_{10} = 1.00E-03 nM^{-1}s^{-1}$	$k_{20f} = 2.00E-03 nM^{-1}s^{-1}$
$k_{11f} = 7.00E-03 nM^{-1}s^{-1}$	$k_{20r} = 0.02 s^{-1}$

$k_{21} = 0.0334563416666667 nM^{-1}s^{-1}$	$k_{36r} = 0.0019 s^{-1}$
$k_{22} = 0.0005555555555555556 s^{-1}$	$k_{37} = 4.0E-006 s^{-1}$
$k_{23} = 100 s^{-1}$	$k_{38} = 2.0E-006 s^{-1}$
$k_{24} = 400 s^{-1}$	$k_{39} = 2.0E-007 s^{-1}$
$k_{25} = 0.4 nM^{-1}s^{-1}$	$k_{40} = 2.0E-008 nM^{-1}s^{-1}$
$k_{26} = 5.0E-005 nM^{-1}s^{-1}$	$k_{41} = 1.1E-006 s^{-1}$
$k_{27} = 0.1 nM^{-1}s^{-1}$	$k_{42} = 2.0E-008 nM^{-1}s^{-1}$
$k_{28} = 200 nM^{-1}s^{-1}$	$k_{43} = 2.0E-008 nM^{-1}s^{-1}$
$k_{29} = 2.8E-004 s^{-1}$	$k_{44} = 2.0E-006 s^{-1}$
$k_{30} = 2.8E-003 s^{-1}$	$k_{45} = 2.0E-006 s^{-1}$
$k_{31} = 0.071 s^{-1}$	$k_{46} = 1.0E-005 nM^{-1}s^{-1}$
$k_{32} = 0.71 s^{-1}$	$k_{47} = 6.0E-012 nM^{-1}s^{-1}$
$k_{33} = 0.2 s^{-1}$	$k_{48} = 3.0E-008 nM^{-1}s^{-1}$
$k_{34} = 0.04 s^{-1}$	$k_{49} = 3.0E-009 s^{-1}$
$k_{35} = 0.033 s^{-1}$	$k_{50} = 1.0E-007 s^{-1}$
$k_{36f} = 0.002 s^{-1}$	$k_{51} = 2.0E-006 s^{-1}$

Transforming state-based models to P Systems models in practice

Petros Kefalas[1], Ioanna Stamatopoulou[2], George Eleftherakis[1], Marian Gheorghe[3]

[1]CITY College, Department of Computer Science,
13 Tsimiski Str., 54624, Thessaloniki, Greece
{kefalas, eleftherakis}@city.academic.gr

[2]South-East European Research Centre,
17 Mitropoleos Str., 54624, Thessaloniki, Greece
istamatopoulou@seerc.org

[3]University of Sheffield, Department of Computer Science,
211 Portobello Str, S14DP, UK
M.Gheorghe@dcs.shef.ac.uk

We present an automatic practical transformation of Communicating X-machines to Population P Systems. The resulting compiler is able to take as input a Communicating X-machine model written in an appropriately designed language (XMDL) and produce a Population P System in another notation (PPSDL). The latter contains only transformation and communication rules. However, the user can further enhance the models with more rules that deal with the reconfiguration of structure of the network of cells. XMDL, PPSDL and their accompanied compilers and animators are briefly presented. The principles of transformations and the transformation templates of the compiler are discussed. We use an example model of a biological system, namely an ant colony, to demonstrate the usefulness of this approach.

1 Introduction

State-based methods, such as finite state machines and their counterparts, are widely used for modelling reactive systems [7]. In particular, X-machines (XMs) possess an intuitive modelling style since they reduce the number of the model's states due to their associated memory structure and they are directly linked to implementation due to transition functions between states. Most importantly, however, X-machines are coupled with techniques for formal verification and testing, reassuring correctness of implementation with respect to their models [4, 2]. There exist several tools that facilitate modelling with X-machines as well as the testing and verification of models [5, 16]. In addition, X-machine models can communicate, thus forming larger scale systems.

Communicating X-machines (CXMs) provide the necessary modelling message passing means and computation that demonstrate the feasibility of scaling up models [8]. However, they do suffer from a major drawback: the organisational structure of the composed system is predefined and remains static throughout the computation. Although for some systems this is a virtue, for some others, such as multi-agent systems, reorganisation is an important feature that should be addressed in a model. In this context, by reorganisation we mean change in the network of communication between agents and change in the number of agents that participate in the system.

Membrane computing, on the other hand, is a relatively new area and its usefulness regarding the modelling of systems has only recently started to be explored. P Systems, however, possess such features that may potentially address the problems stated above [11]. Some initial studies demonstrated that P Systems and its variants, such as Population P Systems (PPSs), could be used to model multi-agent systems [12]. They may not seem as intuitive with respect to modelling behaviours of agents because simple objects and rewriting rules over those objects are not sufficient. But they do deal with reorganisation quite effectively. Rules for division and differentiation of cells as well as cell death and bond-making rules allow for a powerful manipulation of the structure of a multi-agent system and the communication links between agents-cells. Tools have been developed, although not targeted to multi-agent systems [15]. The majority of the tools focus on computation with the rest dealing with some modelling aspects.

A brief comparison between Communicating X-machines and Population P Systems is shown in Table 1.1. Their complementarity has led to the successful integration of the two methods [14].

Modelling feature	CXMs	PPSs
Agent internal state representation	√	
Complex data structures for knowledge, messages, stimuli etc.	√	
Direct communication / Message exchange	√	
Non-deterministic communication		√
Dynamic addition and removal of agent instances		√
Dynamic communications network restructuring		√
Synchronous computation	√	√
Asynchronous computation	√	√
Formal verification of individual components	√	
Test cases generation for individual components	√	
Tool support	√	√

Table 1.1 Comparison of features of X-machines and Population P Systems with respect to modelling.

In this paper, we take a different approach. We attempt to transform existing Communicating X-machine models to Population P Systems models. The transformation is based on the theoretical principles reported in [10]. Here, we deal with the transformation in practice, that is, having a CXM model described in some language for CXMs, we describe the automatic compilation to an equivalent PPS model described in some other language for PPSs. These languages, namely XMDL and PPSDL (DL stands for Description Language) have been developed separately in time, with about a 6-year difference. However, the younger language PPSDL has been influenced by the successful launch and experience we acquired through the use of XMDL. This admittedly eased the implementation of the compiler that does the transformation to some extent.

The rationale behind the transformation is rather simple but we believe an important one. As modellers, we would rarely use PPS for modelling the behaviour and communication between agents. The main reason for that would be the lack of expressive power, as X-machines serve this need in a far better way. So, taking CXM models, which can be individually verified, and transforming them into PPS models we could use PPS rules to extend the model with dynamic features. Practically, this means that not only we surpass the shortcomings of PPS in modelling agent behaviours, but we also feel quite confident (depending on a formal proof that the transformation is correct) that the resulting PPS model meets at least some quality requirements.

The paper is organised as follows. Section 2 is a brief introduction to Communicating X-machines with main focus on XMDL. Section 3 does a similar but slightly more extended introduction to PPSDL. The principles of transformation are briefly listed in Section 4 together with the actual transformation templates from XMDL to PPSDL. We use a simple example, a system of communicating ants, as part of an ant colony, to show the equivalence between the input XMDL and the output PPSDL models. Finally, we discuss certain arising issues and we conclude with directions for future work and extensions.

2 X-Machine Description Language

X-machines are finite state machines with two prominent characteristics; they have an associated memory structure, m, that can hold data and instead of having simple inputs as labels, transitions are triggered through functions, φ, which are activated by inputs, σ, and memory values and produce outputs, γ, while updating the memory values. Of particular interest are stream X-machines which have been extensively used for modelling reactive systems. The formal definition of stream XMs can be found in [4]. An informal but comprehensive abstract model of an XM is depicted in Fig. 2.1.

XM models can communicate by sending messages one to another. There are many alternative definitions of Communicating X-machines. We use a practical approach in which the output of a function of one XM is forwarded as input to a function of another XM [8]. However, in order for this to happen, a requirement should be met: the message

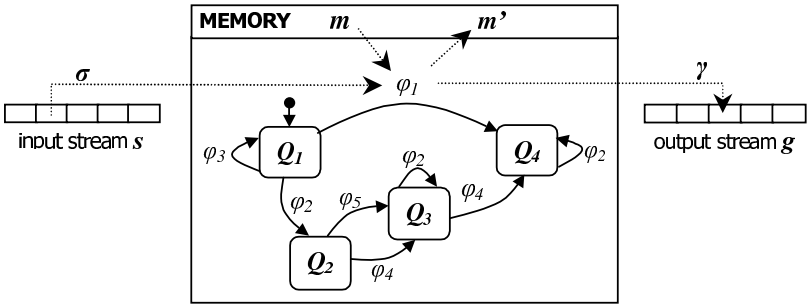


Fig. 2.1 An abstract X-machine.

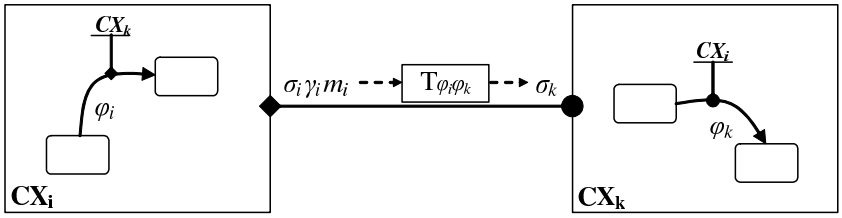


Fig. 2.2 An abstract example of two Communicating X-machines.

sent should be compliant with the input alphabet of the receiving XM. This is why a transformation function, T, is required to transform the message of the sender into an input for the receiver. The concept is shown in Fig. 2.2 while a complete definition of CXMs can be found in [8].

The X-Machine Description Language (XMDL) was developed to assist with the modelling and animation of models [5]. The idea behind XMDL was to use a simple, yet powerful, declarative notation which would be close to the mathematical, yet practical, notation for XMs. XMDL possesses constructs with which one can define an input and an output alphabet set, a memory structure including an initial memory, a set of states including an initial state, transitions between states and functions. Functions get an input and a memory and give an output and a new memory, if certain conditions (guards) hold on input or memory values. The modeller can define any kind of different types of values by combining built-in types, such as natural numbers, with user-defined types, such as sets, sequences, tuples, etc. Table 2.2 informally presents XMDL constructs with a brief explanation on each one. The complete formal XMDL grammar is available from [6].

XMDL has been extended to provide the ability to define the transformation function T, i.e. the function that transforms the output of a CXM to an input of another CXM in a communicating system. XMDL-c also provides the constructs to define instances of class XMs with different initial state and memory as well as communication links between functions of participating CXMs (see Table 2.3).

XM	XMDL_syntax	Informal semantics
Σ	`#input (` $i_1, \ldots, i_n$ `)`	Defines the input tuple for functions
Γ	`#output (` $o_1, \ldots, o_k$ `)`	Defines the output tuple for functions
Q	`#states = {` $q_1, \ldots, q_m$ `}`	Defines the set of states of the XM
M	`#memory (` $m_1, \ldots, m_j$ `)`	Defines the memory tuple of the XM
q_0	`#init_state (` s_0 `)`	Defines the initial state
m_0	`#init_memory (` $v_1, \ldots, v_j$ `)`	Defines the initial memory values
F	`#transition (` q_i, φ_k `) =` q_j	A set of statements that define the transition between states and their corresponding labels (functions)
Φ	`#fun` *name* `(` *input_tuple, memory_tuple* `) =` `if` $condition_1$ `(and|or)` $condition_2 \ldots$ `then (` *output_tuple, memory_tuple* `)` `where` *informative_expression*`.`	Defines a function φ in Φ
	`#type` *identifier* `=` *user defined* `|` *set operations* `|` *built-in type* `|` *tuple*	Defines types of values to be used in all constructs. User defined types include enumerated sets, sequences, etc., while operation include unions, Cartesian products (tuples) etc.

Table 2.2 Main constructs of XMDL (words in upright font are XMDL keywords)

XMDL-c syntax	Informal semantics
`#model` *instance name* `instance_of` *modelname* `with:` `#init_state =` *initial_state*`;` `#init_memory =` *initial_memory_tuple*	Defines an instance of a CXM component with an initial state and initial memory.
`#communication of` *receiver function* `reads from` *sender*`.`	Defines that a function of a receiver XM reads an input from another XM.
`#communication of` *sender function* `writes` *message* `to` *receiver*`.` `using` *variables_in_message* `from (memory|input|output)` *tuple*	Defines T, i.e. the message format, the function of the sender XM and the receiver.

Table 2.3 Main constructs of XMDL-c (words in upright font are XMDL keywords).

A tool, called X-System, has also been implemented [9]. It includes a DCG (Definite Clause Grammar) parser, a syntax and logical error checker, a compiler of XMDL to Prolog and an animator. Models written in XMDL are compiled and animated, that is, the synchronous computation of the CXM model is imitated through inputs provided by the user. Part of X-System's architecture is shown in Fig. 2.3.

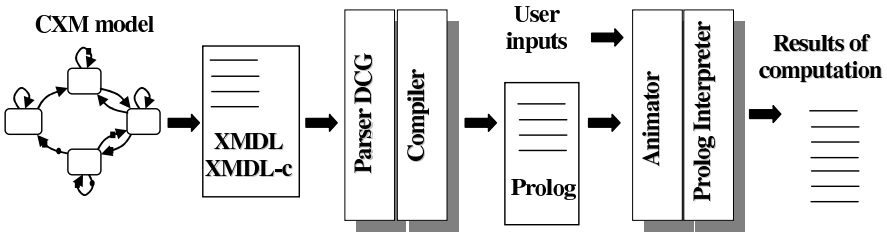


Fig. 2.3 The X-System

3 Population P Systems Description Language

Population P Systems consist of network of cells which are instances of possibly different types. Each type is a class of cells possessing the same rules. Rules consume objects and generate new ones (transformation), in the presence of some objects they change the type of the cell (differentiation), divide the cell (division) or dissolve the cell (death). Communication rules, import or export objects from and to the environments or other neighbouring cells. The latter are determined through bond-making rules that create links between cells if certain conditions hold. The formal definition of PPS can be found in [1]. An abstract PPS model is shown in Fig. 3.4.

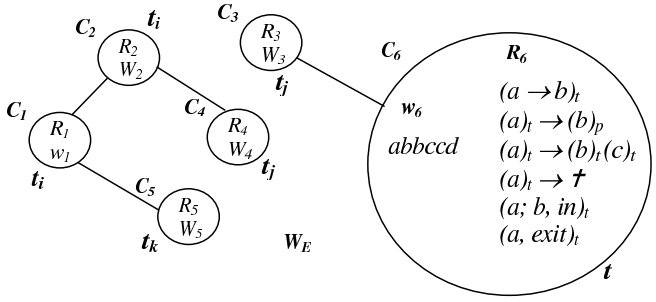


Fig. 3.4 An abstract Population P System.

Similarly to XMDL, Population P Systems Description Language (PPSDL) has been designed so as to allow the experimentation with some PPS models [13]. We decided to keep the concept and, occasionally, the look of XMDL to some extent, and came up with a simple declarative notation, as close as possible to the formal definition of a PPS. PPSDL possesses constructs that allow one to define types of cells, cells as instances of those types, objects in cells and in the environment, as well as all types of rules. What makes PPSDL practical for modelling is the ability to associate objects with types, by combining built-in types with user-defined types. Therefore, objects in cells are characterised by a type identifier; we use the notation `type_identifier*(value)` to denote objects. Table 3.5 informally presents PPSDL constructs with a brief explanation on each one. PPSDL is the core of PPS-System, which includes a DCG parser, a compiler of PPSDL to Prolog and an animator, similar to X-System in Fig. 2.3. The animator

simulates the computation of a PPS model, also allowing the setting of different priorities in rule selection. Briefly, at each cycle, all cells are considered and the applicable rules for each are found and triggered (ordered according to their priority). PPS-System allows the user to input objects directly to cells during computation, if needed, thus allowing more flexibility in the animation.

Cells and types	CXM component	A cell with objects, transformation and communication rules
Objects in Cells	States Q Memory M Inputs Σ Outputs Γ Messages	$(state : q)$, where $q \in Q$ $(memory : m)$, where $m \in M$ $(input : i)$, where $i \in \Sigma$ $(output : o)$, where $o \in \Gamma$ $(message : content)$
Transformation Rules	$\varphi : \Sigma \times M \rightarrow \Gamma \times M$ such that $\varphi(\sigma, m) = (\gamma, m')$, where $m, m' \in M, \sigma \in \Sigma, \gamma \in \Gamma$, for every $q, q' \in Q$ such that $q' \in F(\varphi, q)$	$\varphi_{\sigma,m}$: $((state : q)$ $(memory : m))$ $(input : \sigma)$ $\rightarrow (state : q')$ $(memory : m')$ $(output : \gamma))_t$
Communication Rules	Most general case (incoming and outgoing message) $\varphi : \Sigma \times M \rightarrow \Gamma \times M$ such that $\varphi(\sigma, m) = (\gamma, m')$, where $m, m' \in M, \sigma \in \Sigma, \gamma \in \Gamma$, for every $q, q' \in Q$ such that $q' \in F(\varphi, q)$ and for $incoming \in \Sigma$, $T(\sigma, m, \gamma) = outgoing$	$\varphi_{\sigma,m}$: $((state : q)$ $(memory : m)$ $(message : incoming)$ $\rightarrow (state : q')$ $(memory : m')$ $(output : \gamma)$ $(message : outgoing))_t$

Table 3.4 Principles of transforming a CXM to a PPS.

4 Transforming XMDL to PPSDL

Recently, some basic principles for transforming CXM to PPS have been reported [10] and are briefly presented in Table 3.4. Based on those principles a compiler, which accepts XMDL models and a CXM system formed out of these models and produces PPSDL code for the equivalent PPS, has been developed.

The compiler is written in Prolog (as X-System and PPS-System), based on a set of templates that implement the theoretical transformations. For example, the template compiling the cell types in PPSDL is:

```
if Types = { T | xmdl_c_code=[#model _ instance_of T|_] }
then ppsdl_code = [#cell_types = Types]
```

and the template for compiling a function (with no communication annotation) to a transformation rule in PPSDL is:

```
if xmdl_code = [ #model M ] and
xmdl_code = [ #fun F (I, IM) = If (O, OM) Where ] and
xmdl_code = [ #transition (S1, F) = S2 ] then
ppsdl_code = [ #transformation_rule F
  ( [ state*(S1), memory*(IM), input*(I) ]
  -> [ state*(S2), memory*(OM), output*(O) ] ) : M, If, Where]
```

Sub-statements `If` and `Where` are left untouched, since XMDL and PPSDL share the same syntax for these expressions. As an example, consider an on-off switch which counts how many times its status change. Part of the XMDL code is:

```
#model switch.
#transition (on, switch_off) = off.
#fun switch_off ((turn_off),(?count)) = ((device_off),(?newcount)),
  where ?newCount <- ?count+1.
```

The resulting PPSDL code according to the template above would be:

```
#transformation_rule switch_off
   ([state*(on), memory*((?count)), input*(turn_off)] ->
    [state*(off), memory*((?newcount)), output*(device_off)]):
                                                    switch,
    where ?newCount <- ?count+1.
```

In the case that the CXM function is one that sends a message to another function, a similar transformation rule is produced which also generates an object of the form `outgoing_message*(Message)` in the right hand multiset. An additional communication rule is also created in order to transport this message to the appropriate cell in the form of an `incoming_message*(Message)`. Finally, for a CXM function that reads a message, the object `input*(Input)` is replaced by an object of the form `incoming_message*(Message)` in the left hand side of the corresponding rule.

5 Example transformation: Ants

Consider the case of a Pharaoh ant colony and its behaviour inside the nest (described in more detail in [3]). The ants spend much of their in-nest time doing nothing. Ants doing nothing can be referred to as inactive and can become active, i.e. start searching for food, in the case that its food supplies drop below a particular threshold.

The Pharaoh ants behaviour is presented in a very simplified situation where the colony is situated in an rectangular environment. The ants are either inactive or move around looking for food. When two ants come across they might share food if one is active and the other one is not (in an inactive state). The ants go out to forage when they are hungry, no source food is identified (i.e. no other ant that might provide some food) and a trail pheromone leading to an exit point from the hive is discovered.

The XM model of an ant is depicted in Fig. 5.5 where the • symbol on a function denotes that the function receives input from another machine and, on the contrary, the ⧫ symbol on a function denotes that the function sends a message to be received as input by the function of another machine. This means that communication between two ants is required when they share food using the `giveFood`, and `takeEnoughFood` or `takeNotEnoughFood` functions.

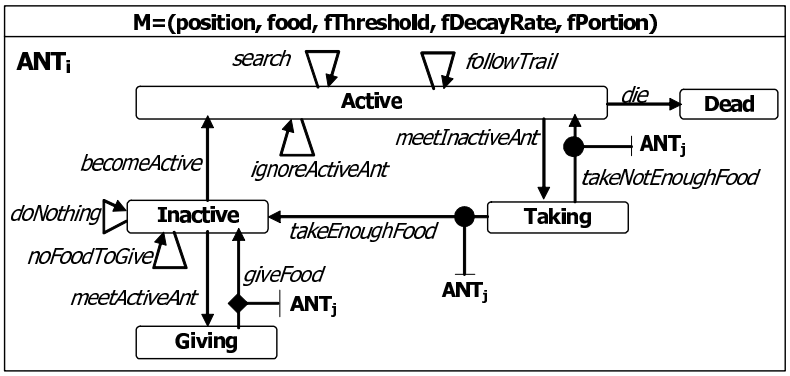


Fig. 5.5 The Pharaoh ant X-machine model.

5.1 Ants in XMDL For demonstration purposes, we consider a colony consisting of two ants (`ant1` and `ant2`) which form a CXM system and need to communicate in order to share food if one of them is inactive and the other is not. Part of the XMDL code modelling a communicating ant is as follows (identifiers preceded with a ? denote variables):

```
#model Ant.

#type coord = natural0.
#type pos = (coord, coord).
#type fPortion = natural.
#type description = {sp, ph, ha, nha}.
#type stimuli = description union fPortion.
#type food = natural0.
#type fThreshold = natural.
#type fDecayRate = natural.
#type outstring = {ignoring_ant, ignoring_hungry_ant, ...,
                                                  giving_food}.

#input (pos, stimuli).
#output (outstring).
#memory (pos, food, fThreshold, fDecayRate, fPortion).
#states = {inactive, hungry, giving, taking, dead}.

#transition (inactive, doNothing) = inactive.
#transition (inactive, noFoodToGive) = inactive.
#transition (inactive, becomeHungry) = hungry.
...
#fun die ((?p, ?in), (?pos, ?f, ?ft, ?fdr, ?mfp)) =
  if ?what_is_left =< 0 then
```

```
  ((dying), (?pos, 0, ?ft, ?fdr, ?mfp))
  where ?what_is_left <- ?f - ?fdr.

#fun giveFood ((?p, ?in), (?pos, ?f, ?ft, ?fdr, ?mfp)) =
  ((giving_food), (?pos, ?nf, ?ft, ?fdr, ?mfp))
  where ?food_reduction <- ?fdr + ?mfp
  and ?nf <- ?f - ?food_reduction.
...

#model ant1 instance_of Ant
  with: #init_state {inactive}; #init_memory ((1,1), 150, 50, 5,
                                                              15).

#communication of function takeEnoughFood:
  #reads from ant2.

#communication of function takeNotEnoughFood:
  #reads from ant2.

#communication of function giveFood:
  #writes (?pos, ?mfp) to (ant2)
  using ?pos from memory (?pos, ?f, ?ft, ?fdr, ?mfp)
  and using ?mfp from memory (?pos, ?f, ?ft, ?fdr, ?mfp).
```

5.2 Ants in PPSDL Running the compiler, we produce the following PPSDL code for the ant system (part only):

```
#define coord = natural0.
...
#define oustring = {ignoring_ant,ignoring_hungry_ant, ...,
                                                    giving_food}.
#define ant_state = {inactive, hungry, giving, taking, dead}.

#object ant_state_object = state*(ant_state).
#object ant_memory_object =
  memory*(pos, food, fthreshold, fdecayrate, fportion).
#object ant_input_object = input*(pos, stimuli).
#object ant_output_object = output*(outstring).

#cell_types = {ant}.
#cell_names = {ant1, ant2}.

#cell ant1 : ( [state*(inactive), memory*((1, 1), 150, 50, 5,
                                              15)], ant).
#cell ant2 : ( [state*(inactive), memory*((1, 1), 150, 50, 5,
                                              15)], ant).

#transformation_rule givefood ( [ state*(giving),
  memory*(?pos,?f,?ft,?fdr,?mfp), input*(?p,?in) ] -> [ state*
                                              (inactive),
  memory*(?pos,?nf,?ft,?fdr,?mfp), output*(giving_food),
```

```
  outgoing_message*(?pos, ?mfp)] ) : ant,
  where ?food_reduction <- ?fdr + ?mfp and ?nf <- ?f - ?
                                                  food_reduction.
...
```

In addition to the above, a communication rule is required to send the message produced by a cell (`outgoing_message*(M)`) to another cell:

```
#communication_exit_rule export_message
  ( when [outgoing_message*(M)] send_to_cell [incoming_message
                                                  *(M)] ):ant.
```

Note that this communication rule also transforms the message object to be exported into `incoming_message*(M)` so that it may be used as input by the receiving cell.

We also add a default bond-making rule of the form:

```
#bond_making_rule export_message
  ( when [ ] : ant and [ ] :ant ).
```

which always creates a communication link between the ants.

The following is an output of the PPS-System animation of the ant colony:

```
----------- CYCLE: 1 ----------------------------------
CELL: ant1 OBJECTS: [[state, inactive],
   [memory, [[1, 1], 150, 50, 5, 15]]] TYPE: ant
CELL: ant2 OBJECTS: [[state, hungry],
   [memory, [[1, 1], 40, 50, 5, 15]]] TYPE: ant
GRAPH = [ (ant1, ant2), (ant2, ant1)]
...
>>> Begin of User Input for all Cells. Provide input...
>>> For cell ant1: [[1,1],ha].
>>> End of User Input for all Cells
...
----------- CYCLE: 2 ----------------------------------
CELL: ant2 OBJECTS: [[state, hungry], [memory,...] TYPE: ant
CELL: ant1 OBJECTS: [[state, inactive], [memory,...],
   [input, [[1, 1], ha]]] TYPE: ant
...
*** Begin Computation cycle for all Cells
*** Computation Cycle for Cell: ant2
*** Computation Cycle for Cell: ant1
----[transformation_rule(meethungryant, ant1, ...]
  Trying to apply Rule : meethungryant
    ant1: buffer-[[state, giving], [memory,...],
      [output, met_hungry_ant]]
*** End of Computation cycle for all Cells
*** Finished with updating Cell Objects with Buffers
```

```
...
>>> For cell ant2: [[1,1],nha].
----------- CYCLE: 3 ----------------------------------
CELL: ant1 OBJECTS: [[state, giving], [memory,...],
   [output, met_hungry_ant]] TYPE: ant
CELL: ant2 OBJECTS: [[state, hungry], [memory,...],
   [input, [[1, 1], nha]]] TYPE: ant

*** Computation Cycle for Cell: ant2
----[transformation_rule(meetnonhungryant, ant2, ... ]
  Trying to apply Rule : meetnonhungryant
    ant2: buffer-[[state, taking], [memory,...],
      [output, found_non_hungry_ant]]
...
>>> For cell ant1: [[1,1],ha].
----------- CYCLE: 4 ----------------------------------
CELL: ant2 OBJECTS: [[state, taking], [memory,...], ...] TYPE:
                                                       ant
CELL: ant1 OBJECTS: [[state, giving], [memory,...],
   [input, [[1, 1], ha]]] TYPE: ant

*** Computation Cycle for Cell: ant1
----[transformation_rule(givefood, ant1, [[state, giving], ...]
  Trying to apply Rule : givefood
    ant1: buffer-[[message, [[1, 1], 15]], [state, inactive],
      [memory,...],...]
----------- CYCLE: 5 ----------------------------------
CELL: ant2 OBJECTS: [[state, taking], [memory,...], ...] TYPE:
                                                       ant
CELL: ant1 OBJECTS: [[message, [[1, 1], 15]], [state, inactive],
   [memory, ...], ...] TYPE: ant
...
*** Computation Cycle for Cell: ant1
----[communication_exit_rule(export_message, ant1, ...]
  Trying to apply Rule : export_message
    ant2: buffer-[[incoming_message, [[1, 1], 15]]]
...
----------- CYCLE: 6 ----------------------------------
CELL: ant2 OBJECTS: [[state, taking], [incoming_message, [[1, 1],
                                                        15]],
   [memory,...],...] TYPE: ant
CELL: ant1 OBJECTS: [[state, inactive], [memory,...], ...] TYPE:
                                                       ant
...
*** Computation Cycle for Cell: ant2
----[transformation_rule(takenotenoughfood, ant2, ...]
  Trying to apply Rule : takenotenoughfood
    ant2: buffer-[[state, hungry], [memory, [[1, 1], 45, 50, 5,
                                                      15]],...]
...
```

The complete specifications of the ants model in XMDL and in PPSDL (as a result

of the transformation process), both languages' manuals as well as the output of the model's animation can be found in [17].

6 Discussion

So far, we presented a transformation of a (static) CXM model to a (static) PPS model. One could enhance the PPS model with features that deal with a potential dynamic structure of the system. For instance:

- if an ant starves to death, it should be removed from the PPS model;
- if another ant becomes part of the system, a new cell should be generated;
- a bond between two cells should be generated only if two ants move to the same position;
- a bond between two cells should cease to exist if two ants are not at the same position.

All the above issues can be dealt with by additional rules of a PPS, such as cell division, cell death, bond-making rules etc. For the first example, a cell death rule such as:

```
#cell_death_rule dr ( [memory*(?pos, 0, ?ft, ?fdr, ?mfp)] ) :
                                                    ant -> +.
```

will do (the ant dies when it has no more food reserves, as indicated by the second memory element). Similarly, a bond-making rule such as:

```
#bond_making_rule neighbours
  ( when [ memory*(?pos, ?f1, ?ft1, ?fdr1, ?mfp1) ] : ant
    and  [ memory*(?pos, ?f2, ?ft2, ?fdr2, ?mfp2) ] : ant ).
```

will produce a bond between two ants in the same position.

Up to date, the transformation of XMDL into PPSDL is almost fully automatic. The only feature that is dealt with manually at the moment is the transformation function T, that is, how the output message of one cell can have the required input format of another cell. For the time being, the compiler provides a template for the message (a tuple of undefined terms) allowing the user to fill in the correct variable names that correspond to the actual content of this message. The reason for this is that there is no obvious way to link XMDL-c source code with the XMDL code of a model and match the variables of the two codes. It is thought that an external function (explicit Prolog code that does the message formation) may solve the problem.

Another improvement that should be made is in the definitions of objects. XMDL code that defines different models might have the same identifiers referring to different types

(e.g. enumerated sets). When these are compiled to PPSDL code, they result in conflicting definitions of the same object. On the contrary, it would be desirable that PPSDL allows encapsulation, that is, facilitates the definition of objects that belong to different types of cells, even with the same identifier. This is left to be designed and implemented in PPSDL in the near future, together with other improvements suggested by the community of researchers who might want to use PPSDL and PPS-System.

7 Conclusions

We presented an automatic transformation from XMDL, a language used to model Communicating X-machines, into PPSDL, a language used to model Population P Systems. The implemented compiler is based on principles of transformation reported in previous work. The benefit we gain from such a transformation is that we take advantage of existing CXM models that have been verified in order to enhance them with features that refer to the dynamic reconfiguration of their structure. Once the CXM model is compiled into a PPS model, one can add more PPS rules that deal with division, differentiation and death of cells. The resulting model can be successfully animated by the PPS-System, which simulates the computation that takes place.

PPS	PPSDL syntax	Informal semantics
V	`#object` (obj_name) `=(`$type\_name_1, \ldots$`)`	Objects in cells defined as either singletons or tuples using only pre-defined types. The union of all objects form the alphabet.
K	`#cell_types =` $(cell\_type_1, \ldots, cell\_type_m)$	Cell types defined as an enumerated set of labels
	`#cell_names =` $\{cell\_name_1, \ldots, cell\_name_n\}$	Cells defined as set of identifiers present in the initial configuration
γ	`#graph` $graph\_name =$ $\{(cell\_name_1, cell\_name_2), \ldots\}$	The initial graph configuration as set of tuples of communicating cells' names
w_E	`#env_objects =` $\{obj_1, obj_2, \ldots\}$	Environment objects defined as an enumerated set of objects
$C_i = (w_i, t_i)$	`#cell` $(cell\_name_1)$ `:` $([obj_1, obj_2, \ldots], cell\_type)$.	Individual cells defined as a tuple of the multiset of objects they contain and their cell type
$(a \rightarrow b)_t$	`#transformation_rule` $rule\_name$ `(`$[obj_1, \ldots]$ `->` $[obj_i, \ldots]$`):`$cell\_type_1$`,` `if` $cond_1$ `(and|or)` $cond_2, \ldots,$ `where` $informative\_expression$`.`	Transformation rule (for a particular cell type)
$(a; b, in)_t$, $(a; b, enter)_t$	`#communication_in_rule` $rule\_name$ `(` `when` $[obj_1, \ldots]$ `receive` $[obj_i, \ldots]$ `(from_cell | from_environment )) :` $cell\_type$`.`	Communication rule (importing objects)
$(a, exit, b)_t$	`#communication_exit_rule` $rule\_name$ `(` `when` $[obj_i, \ldots]$ `(` `send_to_cell` `| send_to_environment)` $[obj_j, \ldots]$ `) :` $cell\_type$`.`	Communication rule (exporting objects)
$(a)_t \rightarrow (b)_p$	`#differentiation_rule` $rule\_name$ `(`$[obj_1, \ldots]$`) :` $cell\_type_1$ `->` `(`$[obj_i, \ldots]$`) :` $cell\_type_2$`.`	Differentiation rule
$(a)_t \rightarrow (b)_t(c)_t$	`#division_rule` $rule\_name$ `(`$[obj_1, \ldots]$`) :` $cell\_type$ `->(`$[obj_i, \ldots]$`) :` $cell\_type$ `(`$[obj_j, \ldots]$`) :` $cell\_type$`.`	Division rule
$(a)_t \rightarrow \dagger$	`#death_rule` $rule\_name$ `(`$[obj_1, \ldots]$`) :` $cell\_type$ `->+`	Death rule
$(i, x_1; x_2, j)$	`#bond_making_rule` $rule\_name$ `(when [`$obj_1, ...$`] :` $cell\_type$ `and [`$obj_1, ...$`] :` $cell\_type$ `).`	Bond-making rule
	`#define` $identifier$ `=` $user\ defined$ `|` $set\ operations$ `|` $built\text{-}in\ type$ `|` $tuple$	Defines types of values to be used in all constructs. User defined types include enumerated sets, sequences, etc., while operation include unions, Cartesian products (tuples) etc.

Table 3.5 Main constructs of PPSDL (words in upright font are PPSDL keywords).

Bibliography

[1] F. Bernardini and M. Gheorghe. Population P Systems. *Journal of Universal Computer Science*, 10(5):509–539, 2004.

[2] G. Eleftherakis. *Formal Verification of X-machine Models: Towards Formal Development of Computer-based Systems*. PhD thesis, Department of Computer Science, University of Sheffield, 2003.

[3] M. Gheorghe, I. Stamatopoulou, M. Holcombe, and P. Kefalas. Modelling dynamically organised colonies of bio-entities. In J.-P. Banâtre, P. Fradet, J.-L. Giavitto, and O. Michel, editors, *Unconventional Programming Paradigms: International Workshop, (UPP'04), Le Mont Saint Michel, France, September 15-17, 2004, Revised Selected and Invited Papers*, volume 3566 of *Lecture Notes in Computer Science*, pages 207–224. Springer-Verlag, 2005.

[4] M. Holcombe and F. Ipate. *Correct Systems: Building a Business Process Solution*. Springer-Verlag, London, 1998.

[5] E. Kapeti and P. Kefalas. A design language and tool for X-machines specification. In D. I. Fotiadis and S. D. Spyropoulos, editors, *Advances in Informatics*, pages 134–145. World Scientific Publishing Company, 2000.

[6] P. Kefalas. *XMDL User Manual*. CITY College, Thessaloniki, Greece – Affiliated Institution of the University of Sheffield, 2000.

[7] P. Kefalas. Formal modelling of reactive agents as an aggregation of simple behaviours. In I. P. Vlahavas and C. D. Spyropoulos, editors, *Proceedings of the 2nd Hellenic Conference on Artificial Intelligence*, volume 2308 of *Lecture Notes in Artificial Intelligence*, pages 461–472. Springer-Verlag, 2002.

[8] P. Kefalas, G. Eleftherakis, and E. Kehris. Modular modelling of large-scale systems using communicating X-machines. In Y. Manolopoulos and S. Evripidou, editors, *Proceedings of the 8th Panhellenic Conference in Informatics*, pages 20–29. Livanis Publishing Company, 2001.

[9] P. Kefalas, G. Eleftherakis, and A. Sotiriadou. Developing tools for formal methods. In *Proceedings of the 9th Panhellenic Conference in Informatics*, pages 625–639, 2003.

[10] P. Kefalas, I. Stamatopoulou, and M. Gheorghe. Principles of transforming Communicating X-machines to Population P Systems. In G. Vaszil, editor, *Proceedings of the International Workshop on Automata for Cellular and Molecular Computing (ACMC'07)*, pages 76–89, 2007.

[11] G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000. Also circulated as a Turku Center for Computer Science (TUCS) report since 1998.

[12] I. Stamatopoulou, M. Gheorghe, and P. Kefalas. Modelling dynamic configuration of biology-inspired multi-agent systems with Communicating X-machines and Population P Systems. In G. Mauri, G. Păun, M. J. Pérez-Jiménez, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing: 5th International Workshop*, volume 3365 of *Lecture Notes in Computer Science*, pages 389–401. Springer-Verlag, Berlin, 2005.

[13] I. Stamatopoulou, P. Kefalas, G. Eleftherakis, and M. Gheorghe. A modelling language and tool for Population P Systems. In *Proceedings of the 10th Panhellenic Conference in Informatics (PCI'05)*, Volos, Greece, November 11-13, 2005.

[14] I. Stamatopoulou, P. Kefalas, and M. Gheorghe. OPERAS: a formal framework for multi-agent systems and its application to swarm-based systems. In A. Artikis, G. O'Hare, K. Stathis, and G. Vouros, editors, *Proceedings of the 8th International Workshop on Engineering Societies in the Agents World (ESAW'07)*, pages 208–223, 2007.

[15] The P Systems webpage. P Systems Software (last visited on April 17, 2008). http://ppage.psystems.eu/index.php/Software.

[16] C. Thomson and M. Holcombe. Using a formal method to model software design in XP projects. In G. Eleftherakis, editor, *Proceedings of the 2nd South-East European Workshop on Formal Methods*, pages 74–88, 2005.

[17] Transforming state-based models to P Systems models in practice. Support documentation and demos. www.city.academic.gr/csd/kefalas/XMDL toPPSDL/index.html.

How Redundant is your Universal Computation Device?

Alberto Leporati, Claudio Zandron, Giancarlo Mauri

Università degli Studi di Milano – Bicocca,
Dipartimento di Informatica, Sistemistica e Comunicazione,
Viale Sarca 336/14, 20126 Milano, Italy
{leporati,zandron,mauri}@disco.unimib.it

Given a computational model $\mathcal{M}$, and a "reasonable" encoding function $\mathcal{C} : \mathcal{M} \to \{0,1\}^*$ that encodes any computation device M of $\mathcal{M}$ as a finite bit string, we define the *description size* $ds_{\mathcal{C}}(M)$ of M (under the encoding $\mathcal{C}$) as the length of $\mathcal{C}(M)$. The description size $ds_{\mathcal{C}}(\mathcal{M})$ of the entire class $\mathcal{M}$ (under the encoding $\mathcal{C}$) can then be defined as the length of the shortest bit string that encodes a *universal* device of $\mathcal{M}$, that is, the minimum of the description sizes $ds_{\mathcal{C}}(M)$ for M that varies over the set of universal devices contained in $\mathcal{M}$. In this paper we compute upper bounds to the description size of three computational models, namely, deterministic register machines, spiking neural P systems and UREM P systems. Then we compute (lower bounds to) the *redundancy* of each of these models, as the ratio between their description size and the description size of a fixed universal deterministic register machine, here taken as a reference. These computations open the way and provide some first partial answers to the following intriguing question: what is the minimal (description) size of a universal computing device?

1 Introduction

As it is well known, the Church–Turing thesis states that every computable function is Turing computable, which means that there exists a (deterministic) Turing machine that computes it. Further, a classic result in the Theory of Computation states that there exists a (deterministic) universal Turing machine U that, given any (deterministic) Turing machine M and its input I, is able to simulate the execution of M with I as input. The simulated machine M has of course to be encoded in an appropriate way to be given as input to U.

However, Turing machines are by no means the only interesting computation devices investigated in the literature: there are, for example, several variants of register (also counter) machines [17, 14], Post's tag systems [21], the lambda calculus [13, 3], several variants of P systems [19, 20, 29], and many others. Confirming the Church–Turing thesis, all these systems can be simulated by deterministic Turing machines; when also

the converse simulation can be performed we say that the corresponding computation device is *Turing complete*, or *universal*, since by simulating a universal Turing machine we are able to compute any Turing computable (that is, partial recursive) function. Indeed, this is what happens with all the computation devices mentioned above, at least in their general (unrestricted) version; indeed, an interesting direction of research in the Theory of Computation consists into putting some constraints to the features which are used by the devices to perform their computations, and see whether they still reach the computational power of (unrestricted) Turing machines.

Looking for small universal computing devices is a natural and well investigated topic in computer science: see e.g. [14, 26, 16, 22–24, 15, 28] for classic computational models, [25, 8, 5] for tissue and symport/antiport P systems, [27, 4] for universal cellular automata such as Conway's Game of Life and Wolfram's Rule 110, and [18] for spiking neural P systems. A related question that we try to answer in this paper is: What is the size of the *smallest* universal computation device? Of course we must agree on the meaning of the term "size"; as we will see in the next section, the size of a given device may depend on several parameters (for example, the number of registers and the number of program instructions when speaking of register machines), whose number and possible values vary depending on the device under consideration. Trying to find a common unit to measure the size of different computation devices, in section 3 we will define the *description size* of a computation device as the number of bits which are needed to describe it. Precisely, for a given model of computation $\mathcal{M}$ (for example, register machines), we will define an encoding function $\mathcal{C} : \mathcal{M} \rightarrow \{0,1\}^*$ that associates a bit string to every computing device M taken from $\mathcal{M}$; the description size $ds_{\mathcal{C}}(M)$ of M (under the encoding $\mathcal{C}$) will be the length of the bit string $\mathcal{C}(M)$, whereas the description size of the entire class $\mathcal{M}$ will be the minimum between the description sizes $ds_{\mathcal{C}}(M)$ for M that varies over the set of *universal* computing devices contained in $\mathcal{M}$. In this paper we start a quest for the shortest possible bit string that describes a universal computation device; in other words, we look for a computational model $\mathcal{M}$ and an encoding function $\mathcal{C} : \mathcal{M} \rightarrow \{0,1\}^*$ such that $\mathcal{M}$ contains at least one universal computation device and $ds_{\mathcal{C}}(\mathcal{M})$ is as low as possible. To this aim, we will compute the description size of randomly generated deterministic register machines, spiking neural P systems [11] and UREM P systems [6], as well as the size of a specific *small* universal instance of each of these computational models. Then, by taking deterministic register machines as the reference model, we will compute the redundancy of the two membrane computing devices here considered as the ratio between their description size and the description size of the smallest (to the best knowledge of the authors) universal deterministic register machine currently known. As a result, we will have an idea about how verbose are such models of computation in catching the notion of universality.

A word of caution is due: with our work, we are not saying that the most compact computational model is the best: a computation device that requires a lot of features to perform its computations may be more interesting than others because of many reasons. A notable example is given by traditional P systems, whose structure and behavior are

inspired from the functioning of living cells; the amount of theoretical results and applications reported in the bibliography of [29] is certainly an indication of how interesting is such a model of computation.

The paper is structured as follows. In section 2 we briefly recall the definition of the computational models we will work upon: deterministic register machines, spiking neural P systems and UREM P systems. In section 3 we will define and compute the description size of *randomly chosen* instances of all these models. We will also consider a universal instance (taken from the literature) of each of these models, and we will compute both its description size and the redundancy with respect to the smallest (to the best knowledge of the authors) currently known deterministic register machine. In section 4 we draw some conclusions, we criticize some limits of our approach and we propose some directions for future research.

2 Some (universal) models of computation

Let us briefly recall the three computational models that we will use to investigate the description size and the redundancy of universal computing devices.

2.1 Deterministic register machines A *deterministic n–register machine* is a construct $M = (n, P, m)$, where $n > 0$ is the number of registers, P is a finite sequence of instructions bijectively labelled with the elements of the set $\{0, 1, \ldots, m-1\}$, 0 is the label of the first instruction to be executed, and $m-1$ is the label of the last instruction of P. Registers contain non-negative integer values. The instructions of P have the following forms:

- $j : (INC(r), k)$, with $j, k \in \{0, 1, \ldots, m-1\}$ and $r \in \{0, 1, \ldots, n-1\}$
  This instruction, labelled with j, increments the value contained in register r, and then jumps to instruction k.
- $j : (DEC(r), k, l)$, with $j, k, l \in \{0, 1, \ldots, m-1\}$ and $r \in \{0, 1, \ldots, n-1\}$
  If the value contained in register r is positive then decrement it and jump to instruction k. If the value of r is zero then jump to instruction l (without altering the contents of the register).

Computations start by executing the first instruction of P (labelled with 0), and terminate when the instruction currently executed tries to jump to label m.

Formally, a *configuration* is a $(n+1)$-tuple whose components are the contents of the n registers, and the label of the next instruction of P to be executed. An *initial configuration* is a configuration in which the input values $(n_1, \ldots, n_\alpha) \in \mathbb{N}^\alpha$ are stored in registers 0 to $\alpha - 1$, all the other registers are set to 0, and the label of the next istruction to be executed is also set to 0. A *final configuration* is a configuration in which the label of the next instruction is equal to m. A *computation* starts in the initial configuration,

and proceeds by performing computation steps. A *computation step* consists of executing the current instruction, modifying accordingly the contents of the affected register and the pointer to the next instruction to be executed. A computation halts if it reaches a final configuration. The contents of the registers in the final configuration are regarded as the result of the computation. A non–halting computation does not produce a result.

Register machines provide a simple universal computational model. Indeed, the results proved in [7] (based on the results established in [17]) as well as in [9] and [10] immediately lead to the following proposition.

Proposition 1 *For any partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$ there exists a deterministic $(\max\{\alpha, \beta\} + 2)$–register machine M computing f in such a way that, when starting with $(n_1, \ldots, n_\alpha) \in \mathbb{N}^\alpha$ in registers 0 to $\alpha - 1$, M has computed $f(n_1, \ldots, n_\alpha) = (r_1, \ldots, r_\beta)$ if it halts in the final label m with registers 0 to $\beta - 1$ containing r_1 to r_β, and all other registers being empty; if the final label cannot be reached, then $f(n_1, \ldots, n_\alpha)$ remains undefined.*

2.2 Spiking neural P systems Spiking neural P systems (SN P systems, for short) have been introduced in [11] as a class of synchronous, parallel and distributed computing devices, inspired by the neurophysiological behavior of neurons sending electrical impulses along axons to other neurons.

In SN P systems the cells (also called *neurons*) are placed in the nodes of a directed graph, called the *synapse graph*. The contents of each neuron consist of a number of copies of a single object type, called the *spike*. Every cell may also contain a number of *firing* and *forgetting* rules. Firing rules allow a neuron to send information to other neurons in the form of spikes, which are accumulated at the target cells. The applicability of each rule is determined by checking the contents of the neuron against a regular set associated with the rule. In each time unit, if a neuron can use some of its rules then one of such rules must be used. The rule to be applied is nondeterministically chosen. Thus, the rules are used in a sequential manner in each neuron, but neurons function in parallel with each other. Observe that, as usually happens in membrane computing, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized. When a cell sends out spikes it becomes "closed" (inactive) for a specified period of time, that reflects the refractory period of biological neurons. During this period, the neuron does not accept new inputs and cannot "fire" (that is, emit spikes). Another important feature of biological neurons is that the length of the axon may cause a time delay before a spike reaches its target. In SN P systems this delay is modeled by associating a delay parameter to each rule which occurs in the system. If no firing rule can be applied in a neuron, there may be the possibility to apply a *forgetting rule*, that removes from the neuron a predefined number of spikes.

Formally, a *spiking neural membrane system* (SN P system, for short) of degree $m \geq 1$, as defined in [12] in the computing version (i.e., able to take an input and provide and

output), is a construct of the form

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, where:
   (a) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
   (b) R_i is a finite set of *rules* of the following two forms:
      (1) *firing* (also *spiking)* rules $E/a^c \rightarrow a; d$, where E is a regular expression over a, and $c \geq 1$, $d \geq 0$ are integer numbers; if $E = a^c$, then it is usually written in the simplified form: $a^c \rightarrow a; d$; similarly, if $d = 0$ then it can be omitted when writing the rule;
      (2) *forgetting* rules $a^s \rightarrow \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a; d$ of type (1) from R_i, we have $a^s \notin L(E)$ (the regular language defined by E);
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$, with $(i, i) \notin syn$ for $1 \leq i \leq m$, is the directed graph of *synapses* between neurons;
4. $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and the *output* neurons of Π, respectively.

A firing rule $E/a^c \rightarrow a; d \in R_i$ can be applied in neuron σ_i if it contains $k \geq c$ spikes, and $a^k \in L(E)$. The execution of this rule removes c spikes from σ_i (thus leaving $k - c$ spikes), and prepares one spike to be delivered to all the neurons σ_j such that $(i, j) \in syn$. If $d = 0$ then the spike is immediately emitted, otherwise it is emitted after d computation steps of the system. As stated above, during these d computation steps the neuron is *closed*, and it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send a spike along it, then that particular spike is lost), and cannot fire (and even select) rules. A *forgetting* rule $a^s \rightarrow \lambda$ can be applied in neuron σ_i if it contains *exactly* s spikes; the execution of this rule simply removes all the s spikes from σ_i.

A common generalization of firing rules was introduced in [2, 18] under the name of *extended rules*. These rules are of the form $E/a^c \rightarrow a^p; d$, where $c \geq 1$, $p \geq 1$ and $d \geq 0$ are integer numbers. The semantics of these rules is the same as above, with the difference that now p spikes are delivered (after d time steps) to all neighboring neurons.

The *initial configuration* of the system is described by the numbers $n_1, n_2, \ldots, n_m$ of spikes present in each neuron, with all neurons being open. During the computation, a configuration is described by both the contents of each neuron and its *state*, which can be expressed as the number of steps to wait until it becomes open (zero if the neuron is already open).

A *computation* starts in the initial configuration. In order to compute a function $f : \mathbb{N} \to \mathbb{N}$ (functions of the kind $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$, for any fixed pair of integers $\alpha \geq 1$ and $\beta \geq 1$, can also be computed by using appropriate bijections from $\mathbb{N}^\alpha$ and $\mathbb{N}^\beta$ to $\mathbb{N}$), a positive integer number is assigned as input to the specified input neuron in. In the original model, as well as in some early variants, the number is encoded as the number of time steps elapsed between the insertion of two spikes into the neuron. To pass from a configuration to another one, for each neuron a rule is chosen among the set of applicable rules, and is executed. Generally, a computation may not halt. However, in any case the output of the system is considered to be the time elapsed between the arrival of two spikes in the designated output cell out. Other possibilities exist to encode input and output numbers, as discussed in [12]: as the number of spikes contained in neuron in (resp., out) at the beginning (resp., the end) of the computation, as the number of spikes fired in a given interval of time, etc.

In [11] it is shown that *generative* (nondeterministic) SN P systems are universal, that is, can generate any recursively enumerable set of natural numbers. Moreover, a characterization of semilinear sets is obtained by spiking neural P systems with a bounded number of spikes in the neurons. In [18] two small deterministic universal SN P systems are defined, one containing only standard rules, with 84 neurons, and one containing extended rules, with 49 neurons. Their universality is proved by simulating deterministic register machines, by using appropriate modules to read the input spike train, to simulate INC and DEC instructions, and to produce (if and when the computation of the simulated register machine halts) the output spike train.

2.3 UREM P systems P systems with unit rules and energy assigned to the membranes (UREM P systems, for short) have been introduced in [6] as a variant of P systems in which a non–negative integer value (regarded as an amount of energy) is assigned to each membrane of the system. The rules are assigned to the membranes rather than to the regions of the system, and operate like filters that control the movement of objects (symbols of an alphabet) across the membranes.

Formally, a UREM P system [6] of degree $d + 1$ is a construct Π of the form:

$$\Pi = (A, \mu, e_0, \ldots, e_d, w_0, \ldots, w_d, R_0, \ldots, R_d)$$

where:

- A is an alphabet of *objects*;
- μ is a *membrane structure*, with the membranes labelled by numbers $0, \ldots, d$ in a one-to-one manner;
- $e_0, \ldots, e_d$ are the initial energy values assigned to the membranes $0, \ldots, d$. In what follows we assume that $e_0, \ldots, e_d$ are non–negative integers;
- $w_0, \ldots, w_d$ are multisets over A associated with the regions $0, \ldots, d$ of μ;
- $R_0, \ldots, R_d$ are finite sets of *unit rules* associated with the membranes $0, \ldots, d$. Each rule has the form $(\alpha : a, \Delta e, b)$, where $\alpha \in \{in, out\}$, $a, b \in A$, and $|\Delta e|$ is

the amount of energy that — for $\Delta e \geq 0$ — is added to or — for $\Delta e < 0$ — is subtracted from e_i (the energy assigned to membrane i) by the application of the rule.

By writing $(\alpha_i : a, \Delta e, b)$ instead of $(\alpha : a, \Delta e, b) \in R_i$, we can specify only one set of rules R with $R = \{(\alpha_i : a, \Delta e, b) \ : \ (\alpha : a, \Delta e, b) \in R_i, \ 0 \leq i \leq d\}$.

The *initial configuration* of Π consists of $e_0, \ldots, e_d$ and $w_0, \ldots, w_d$. The transition from a configuration to another one is performed by *non-deterministically* choosing one rule from some R_i and applying it (observe that here we consider a *sequential* model of applying the rules instead of choosing rules in a maximally parallel way, as it is often required in P systems). Applying $(in_i : a, \Delta e, b)$ means that an object a (being in the membrane immediately outside of i) is changed into b while entering membrane i, thereby changing the energy value e_i of membrane i by Δe. On the other hand, the application of a rule $(out_i : a, \Delta e, b)$ changes object a into b while leaving membrane i, and changes the energy value e_i by Δe. The rules can be applied only if the amount e_i of energy assigned to membrane i fulfills the requirement $e_i + \Delta e \geq 0$. Moreover, we use some sort of local priorities: if there are two or more applicable rules in membrane i, then one of the rules with $\max |\Delta e|$ has to be used.

A *computation* is a sequence of transitions; it is *successful* if and only if it halts. The *result* of a successful computation is considered to be the distribution of energies among the membranes (a non–halting computation does not produce a result). If we consider the energy distribution of the membrane structure as the input to be processed and the resulting output, we obtain a model for computing functions of the kind $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$, for any fixed pair of integers $\alpha, \beta \geq 1$. The following result, proved in [6], establishes computational completeness for this model of P systems.

Proposition 2 *Every partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$ can be computed by a UREM P system with (at most)* $\max\{\alpha, \beta\} + 3$ *membranes.*

The proof of this proposition is obtained by simulating deterministic register machines. In the simulation, a P system is defined which contains one elementary membrane into the skin for each register of the simulated machine. The contents of the register are expressed as the energy value e_i assigned to the i-th subsystem. A single object is present in the system at every computation step, which stores the label of the instruction of the program P currently simulated. Increment instructions of the kind $j : (INC(i), k)$ are simulated in two steps by using the rules $(in_i : p_j, 1, \widetilde{p_j})$ and $(out_i : \widetilde{p_j}, 0, p_k)$. Decrement instructions of the kind $j : (DEC(i), k, l)$ are also simulated in two steps, by using the rules $(in_i : p_j, 0, \widetilde{p_j})$ and $(out_i : \widetilde{p_j}, -1, p_k)$ or $(out_i : \widetilde{p_j}, 0, p_l)$. The use of priorities associated to these last rules is crucial to correctly simulate a decrement instruction. For the details of the proof we refer the reader to [6].

We conclude this section by observing that if we just want to reach Turing completeness with UREM P systems it is not necessary to exploit neither nondeterminism nor the membrane structure: as stated above, a simple *deterministic* system which is composed by a skin that contains an elementary membrane for each register of the simulated register machine suffices. It will be thus interesting to measure how redundant is this universal model of computation with respect to the more compact deterministic register machines. We also observe that the use of local priorities is instead important, since by omitting them we do not get systems with universal computational power. Precisely, in [6] it is proved that (nondeterministic) UREM P systems without priorities and with an arbitrary number of membranes characterize the family $PsMAT^{\lambda}$ of Parikh sets generated by context–free matrix grammars (without occurrence checking and with λ-rules).

3 Description size

As stated in the Introduction, we define the *description size* of a given computation device M as the length of the binary string which encodes the structure of M. Since this is an informal definition, we have to discuss some technical difficulties that immediately arise. First of all, for any given computational model $\mathcal{M}$ (register machines, SN P systems, etc.) we have to find a "reasonable" encoding function $\mathcal{C} : \mathcal{M} \to \{0,1\}^*$, in the sense given in [1]. Such a function should be able to encode *any* computation device M of $\mathcal{M}$ as a finite bit string. When this string is interpreted (that is, decoded) according to a specified set of rules (the *decoding algorithm*), the decoder unambiguously recovers the structure of M. In order to avoid cheating — by hiding information into the encoding or decoding algorithms — we ask to consider only reasonable encodings that satisfy the following requirements.

1. For each model of computation, the encoding and decoding algorithms are fixed a priori, and their representation as a program for a deterministic register machine or as a deterministic Turing machine have a fixed *finite length*. Note that, when computing the description size of a given device, we will not count the size of the encoding and decoding algorithms; moreover, instead of formally specifying such algorithms, we will only provide *informal* instructions on how to encode and decode our computation devices. An alternative approach, not followed in this paper, consists of minimizing the size of the decoding — and, possibly, encoding — algorithm (that is, its Kolmogorov complexity) together with the length of the encoded strings.
2. With the selected encoding algorithm it should be possible to describe *any* instance of the computational model under consideration (for example, any deterministic register machine). Encodings that allow to represent in a very compact form only one or a few selected instances of the computational model (for example, all the register machines whose program P contains exactly five instructions) are not considered acceptable.

Another point to consider is the following: Given an encoding function for a computational model, how much should we care about the fact that it produces the shortest possible bit strings? On the one hand, we clearly prefer bit strings which are as short as possible. On the other hand, without loss of generality we can restrict our attention to encodings that produce strings whose length is polynomial with respect to the parameters that determine the size of the encoded device (for example, the number n of registers and the length m of the program P in deterministic register machines). All such encodings produce strings whose lengths differ by at most a polynomial amount, and thus we could be tempted to accept such differences and decide to work with handy but verbose encodings rather than with more compact but also uncomfortable ones. In order to find results which depend as little as possible on the chosen encoding, when calculating the description size of our universal instances of the computational models here considered (deterministic register machines, SN P systems and UREM P systems) we will use (when feasible) the *entropy* of the encoded string to determine its description size. This behavior is justified by the fact that optimal entropy-based lossless compressions, such as the Huffman encoding, produce binary strings whose average length is as near as possible to the entropy (that is, the average amount of information) of the uncompressed input string. Of course many other lossless compression algorithms may be used, but we will not consider them in this paper.

We can look at any computational model as a family of computation devices, whose size depends upon a prefixed collection of parameters. For example, the class of all deterministic register machines is composed by machines which have n registers and whose programs are composed by m instructions, for all possible integers $n \geq 1$ and $m \geq 0$. Denoted by $\mathcal{M}$ a computational model, and by $(n_1, n_2, \ldots, n_k)$ the non–negative integer parameters upon which the size of the computing devices of $\mathcal{M}$ depend, we can write $\mathcal{M} = \bigcup_{n_1,\ldots,n_k} \{M(n_1, \ldots, n_k)\}$, where $M(n_1, \ldots, n_k)$ is the instance of $\mathcal{M}$ (that is, a specific computation device) for which the parameters have the indicated values $n_1, \ldots, n_k$.

Let $\mathcal{C} : \mathcal{M} \to \{0,1\}^*$ denote a fixed encoding of $\mathcal{M}$, and let M be a computation device from $\mathcal{M}$. By $ds_{\mathcal{C}}(M) = |\mathcal{C}(M)|$ (the length of $\mathcal{C}(M)$) we will denote the description size of M, obtained by using the encoding $\mathcal{C}$, and by $ds_{\mathcal{C}}(\mathcal{M})$ we will denote the length of the most compact representation — produced by the encoding algorithm of $\mathcal{C}$ — of a *universal* computing device taken from the class $\mathcal{M}$, that is:

$$ds_{\mathcal{C}}(\mathcal{M}) = \min\{ds_{\mathcal{C}}(M) \ : \ M \in \mathcal{M} \text{ is universal}\}$$

By definition, for any fixed universal computing device $M \in \mathcal{M}$ the value $ds_{\mathcal{C}}(M)$ is an upper bound for $ds_{\mathcal{C}}(\mathcal{M})$. We say that a universal computing device $M^* \in \mathcal{M}$ is *optimal* (referred to the description size, for a prefixed encoding $\mathcal{C}$) if $ds_{\mathcal{C}}(M^*) = ds_{\mathcal{C}}(\mathcal{M})$. Given two classes of computational devices $\mathcal{M}$ and $\mathcal{M}'$ (with possibly $\mathcal{M} = M'$), we define the *redundancy* of a universal computation device $M' \in \mathcal{M}'$, with respect to the computational model $\mathcal{M}$ and the encoding $\mathcal{C}$, as $R_{\mathcal{M},C}(M') = \frac{ds_{\mathcal{C}}(M')}{ds_{\mathcal{C}}(\mathcal{M})}$. Similarly, we define the redundancy of a computational model $\mathcal{M}'$ (with respect to $\mathcal{M}$ and $\mathcal{C}$) as

$R_{\mathcal{M},C}(\mathcal{M}') = \frac{ds_{\mathcal{C}}(\mathcal{M}')}{ds_{\mathcal{C}}(\mathcal{M})}$. Finally, by letting $\mathcal{C}$ vary on the class of all possible "reasonable" encodings, for any computational models $\mathcal{M}$ and $\mathcal{M}'$ we can define:

$$ds(\mathcal{M}) = \min_{\mathcal{C}}\{ds_{\mathcal{C}}(\mathcal{M})\} \qquad \text{and} \qquad R_{\mathcal{M}}(\mathcal{M}') = \frac{ds(\mathcal{M}')}{ds(\mathcal{M})}$$

that is, the description complexity of $\mathcal{M}$ and the redundancy of $\mathcal{M}'$ with respect to $\mathcal{M}$, respectively.

Let us note that the quantities $ds(\mathcal{M})$ and $ds_{\mathcal{C}}(\mathcal{M})$, for some fixed computational model $\mathcal{M}$ and encoding $\mathcal{C}$, may be difficult to find, as it usually happens with theoretical bounds. Hence in general we will obtain upper bounds to these quantities, and thus lower bounds for the corresponding redundancies. The final goal of the research line set out with this paper is to find a universal computational class $\mathcal{M}$ and an encoding $\mathcal{C} : \mathcal{M} \to \{0,1\}^*$ whose description size $ds_{\mathcal{C}}(\mathcal{M})$ is as low as possible, and eventually an optimal instance $M \in \mathcal{M}$. In this way, no other model $\mathcal{M}'$ that contains a universal computation device would have $ds(\mathcal{M}') < ds_{\mathcal{C}}(\mathcal{M})$, and hence the value $ds_{\mathcal{C}}(M) = ds_{\mathcal{C}}(\mathcal{M})$ could be regarded as the *description size complexity* of *universality*: in other words, it would be the minimal number of bits which are needed to describe the ability to compute Turing computable (that is, partial recursive) functions.

In the next subsections we will compute the description size of *randomly generated* computing devices taken from the classes recalled in section 2: deterministic register machines, spiking neural P systems and UREM P systems. Then, we will also compute the description size of a small universal device taken from each of these classes, with respect to a prefixed encoding, thus providing upper bounds to the description sizes of the whole classes.

In what follows, for any natural number n we will simply denote by $\lg n$ the number $\lceil \log_2 n \rceil + 1$ of bits which are needed to represent n in binary form.

3.4 Deterministic register machines Let DRM denote the class of deterministic register machines. In order to encode a particular machine $M \in \mathrm{DRM}$ we first have to specify the number n of registers (that will be numbered from 0 to $n-1$) and the number m of instructions (numbered from 0 to $m-1$) that compose the program P. These will be the parameters upon which the size of M will depend. Denoted by $\mathrm{DRM}(n,m)$ the subclass of register machines that have n registers and programs composed of m instructions, we can thus write DRM = $\bigcup_{n\geq 1, m\geq 0} \mathrm{DRM}(n,m)$.

Let $M \in \mathrm{DRM}(n,m)$ be a register machine, and let P denote its program. To describe a single instruction of P, we need 1 bit to say whether it is an INC or a DEC instruction, $\lg n$ bits to specify the register which is affected, and $\lg m$ bits (resp., $2 \cdot \lg m$ bit) to specify the jump label (resp., the two jump labels) for the INC (resp., DEC) instruction.

$0 : (DEC(1), 1, 2)$ $1 : (INC(7), 0)$
$2 : (INC(6), 3)$ $3 : (DEC(5), 2, 4)$
$4 : (DEC(6), 5, 3)$ $5 : (INC(5), 6)$
$6 : (DEC(7), 7, 8)$ $7 : (INC(1), 4)$
$8 : (DEC(6), 9, 0)$ $9 : (INC(6), 10)$
$10 : (DEC(4), 0, 11)$ $11 : (DEC(5), 12, 13)$
$12 : (DEC(5), 14, 15)$ $13 : (DEC(2), 18, 19)$
$14 : (DEC(5), 16, 17)$ $15 : (DEC(3), 18, 20)$
$16 : (INC(4), 11)$ $17 : (INC(2), 21)$
$18 : (DEC(4), 0, 22)$ $19 : (DEC(0), 0, 18)$
$20 : (INC(0), 0)$ $21 : (INC(3), 18)$

Fig. 3.1 The small universal deterministic register machine defined in [14]

A simple encoding of a deterministic register machine having n registers and m instructions is a sequence of m blocks, each composed of $1+\lg n+\lg m$ or $1+\lg n+2\cdot\lg m$ bits, encoding the corresponding instruction of P. In order to correctly and unambiguously decode a bit string that encodes a register machine from $\mathrm{DRM}(n, m)$, we also need to store in a separate place the numbers n and m; this will require further $\lg n + \lg m$ bits.

For a *randomly chosen* (and thus, possibly, non–universal) machine $M \in \mathrm{DRM}(n, m)$, about half of the instructions of P will be INCs and half will be DECs; hence the description size of M, with respect to the encoding $\mathcal{C}$ we have just defined, will be:

$$\begin{aligned} ds_{\mathcal{C}}(M) &= \frac{m}{2}\left[1 + \lg n + \lg m\right] + \frac{m}{2}\left[1 + \lg n + 2 \cdot \lg m\right] \\ &= m \cdot \left[1 + \lg n\right] + \frac{3m}{2}\ \lg m \end{aligned}$$

In order to compute an upper bound to $ds(DRM)$ we have instead to restrict our attention to *universal* register machines. In [14], several universal register machines are described and investigated. In particular, the *small* universal register machine depicted here in Figure 3.1 is defined. This machine has $n = 8$ registers and $m = 22$ instructions. However, recall that we need a further label (22) to halt the execution of P anytime by simply jumping to it, and thus we put $m = 23$. The number of bits required to store these values are 3 and 5, respectively. The encoding of this machine produces a bit string which is composed of 22 blocks, one for each instruction of P. Each register will require 3 bits to be specified, and each label will require 5 bits. If we denote INC instructions by a 0, and DEC instructions by a 1, then the first block will be 1 001 00001 00010, where we have put a small space to make clear how the block is

formed: the first 1 denotes a DEC instruction, which has to be applied to register number 1 (= 001), and the two labels to jump to when we have executed the instruction are 1 (= 00001) and 2 (= 00010). Similarly, the block that encodes the second instruction is 011100000 (here we have omitted the unnecessary spaces), whereas the string that encodes the whole machine M is:

10010000100010 011100000 011000011 11010001000100
11100010100011 010100110 11110011101000 000100100
11100100100000 011001010 11000000001011 11010110001101
11010111001111 10101001010011 11011000010001 10111001010100
010001011 001010101 11000000010110 10000000010010
000000000 001110010

Here the spaces denote a separation between two consecutive blocks; of course these spaces are put here only to help the reader, but are not necessary to decode the string. We can thus conclude that, referring to the encoding $\mathcal{C}$ given above:

$$ds_{\mathcal{C}}(M) = 14 * 13 + 9 * 9 = 182 + 81 = 263$$

Since our final goal is to find the shortest bit string that encodes a universal computation device, we could wonder how many bits we would save by *compressing* the above sequence. Many compression algorithms exist, that yield to different results. Here we just consider entropy-based compressors, such as the Huffman algorithm, and we compute a bound on the length of the compressed string. If we look at the above bit string, we can see that it contains 154 zeros and 109 ones. Hence in each position of the string we have the probability $p_0 = \frac{154}{263}$ that a 0 occurs, and the probability $p_1 = \frac{109}{263}$ that a 1 occurs. By looking at the output of the encoding algorithm as a *memoryless* information source, we can compute the entropy of the above sequence, that measures the average amount of information carried by each bit of the string:

$$H(M) = -p_0 \log_2 p_0 - p_1 \log_2 p_1 \approx 0.979$$

Now, by applying an optimal entropy-based compressor we would obtain a compressed string whose length is approximately equal to the length of the uncompressed string times the entropy, that is, $\lceil 263 \cdot 0.979 \rceil = 258$ bits. Such a quantity is less than 263, but of course is still an upper bound to $ds(\mathrm{DRM})$, the (unknown, and possibly very difficult to determine) description size complexity of deterministic register machines.

These results can be compared with those obtained by observing that (of course) also Figure 3.1 depicts an encoding of the register machine: by ignoring the spaces, carriage returns and the labels in front of each instruction, we can represent the machine as $(DEC(1), 2)(INC(7), 0)(INC(6), 3)\ldots$ that, expressing each character in standard 7-bit ASCII code, produces a string of 1841 bits (= $263 \cdot 7$, exactly seven times the length obtained with the previous encoding), containing 1082 zeros and 759 ones. The

probability that a 0 occurs in such a string is thus $p_0 = \frac{1082}{1841} \approx 0.588$; similarly, the probability that a 1 occurs is $p_1 = \frac{759}{1841} \approx 0.418$, and thus the entropy is $H(M) = -p_0 \log_2 p_0 - p_1 \log_2 p_1 \approx 0.978$. As we can see, we have obtained almost exactly the same value for the entropy that we obtained with the previous encoding (indeed, we performed our computation using three decimal digits, hence the third decimal digit of the result could be wrong due to roundings). This time, however, by compressing the encoded string by means of an optimal entropy encoder we would obtain about $\lceil 1841 \cdot 0.978 \rceil = 1801$ bits. This result confirms that the first encoding has to be preferred if one looks for compact representations.

As stated above, the choice of a different category of (lossless) compression algorithms may yield to different string lengths. Moreover, even if we restrict our attention to entropy-based compressors the fact that we have modelled the output of the encoding algorithm as a memoryless information source is questionable. In fact, it is clear that when we encouter a 0 at the beginning of a new block that encodes an instruction of the register machine then we already know that 8 bits will follow, instead of 13, since we are reading an INC instruction. This means that the occurrence of the bits in the sequence does depend somehow upon the bits which have already been emitted by the source, and to capture this dependence we should use a source endowed with memory, such as a Markovian source.

3.5 Spiking neural P systems Let us denote by SNP the class of SN P systems. In order to encode a particular system $\Pi \in \mathrm{SNP}$, we have to specify the following parameters:

- the *degree* m, that is, the number of neurons;
- the total number R of rules contained in the system;
- the maximum number C of spikes consumed by any rule in the system;
- the maximum delay D that occurs in the rules.

In order to describe the synapse graph (which is a directed graph, without self–loops) we need $m^2 - m$ bits. To describe a forgetting rule $a^s \to \lambda$ we need 1 bit to distinguish it from spiking rules, and $\lg C$ bits to represent the value of s. On the other hand, to describe a firing rule $E/a^c \to a; d$ we need 1 bit to distinguish it from forgetting rules, $\lg C$ bits to represent c and $\lg D$ bits to represent d; moreover, we need some bits to describe the regular expression E. In general, there are no limitations to the length of a regular expression, but by observing the systems described in [18] we note that the following five types of expressions suffice to reach computational completeness:

$$a, \quad a^2, \quad a^3, \quad a(aa)^+, \quad a(aa)^*$$

and thus we will restrict our attention to systems that contain only these kinds of regular expressions. Of course this restriction will influence our results; any different choice of the set of regular expressions is valid, provided that the class of SN

P systems thus obtained contains at least one universal computation device (if one is interested in finding an upper bound to the description size of SN P systems). To specify one of the above five expressions we need 3 bits, and hence we need a total of $1 + \lg C$ bits to describe a forgetting rule, and $1 + 3 + \lg C + \lg D$ bits to describe a firing rule. On the average, a *randomly generated* SN P system with R rules will contain about $\frac{R}{2}$ firing rules and $\frac{R}{2}$ forgetting rules, and thus we will need $\frac{R}{2}[1 + \lg C] + \frac{R}{2}[4 + \lg C + \lg D] = R[1 + \lg C] + \frac{R}{2}[3 + \lg D]$ bits to encode them.

A simple encoding of an SN P system $\Pi(m, R, C, D)$ of degree m, having R rules that consume at most C spikes each, and whose delays are less than or equal to D, is a sequence of m blocks — one for each neuron — followed by the $m^2 - m$ bits that encode the structure of the synapse graph. For each neuron, we have to specify the list of its rules; since each neuron may have a different number of rules, we could put an additional bit in front of the encoding of each rule, to signal whether such a rule is the last in the description of the neuron. However, in order to be able to encode also the empty list of rules, we will put a bit equal to 1 in front of each rule, and a 0 at the end of the list. In this way, when decoding, the presence of a 1 means that the next bits encode a rule of the neuron, whereas a 0 means that the next bits encode a different neuron. Using this encoding $\mathcal{C}$, the description size of a *randomly chosen* (and thus, possibly, non–universal) system Π is:

$$ds_{\mathcal{C}}(\Pi) = \frac{R}{2}[2 + \lg C] + \frac{R}{2}[5 + \lg C + \lg D] + R + m^2 - m = \\ = \frac{9R}{2} + R \lg C + \frac{R}{2} \lg D + m^2 - m$$

As we did with register machines, in order to determine an upper bound to the description size $ds(SNP)$ of the entire class of SN P systems we now turn our attention to *universal* systems. In [18], a *small* universal SN P system is obtained by simulating a slightly modified version of the small universal deterministic register machine described in section 3.1. The modification is needed for a technical reason due to the behavior of SN P systems (see [18] for details); the modified version of the register machine has 9 registers, 24 istructions and 25 labels. Each instruction is simulated through an appropriate subsystem; moreover, an INPUT module is needed to read the input spike train from the environment and initialize the simulation, and an OUTPUT module is needed to produce the output spike train if and when the computation of the simulated register machine halts. As a result, the universal SN P system is composed of 91 neurons, which are subsequently reduced to 84 by simulating in a different way one occurrence of two consecutive $INCs$ and two occurrences of an INC followed by a DEC. These 84 neurons are used as follows: 9 neurons for the registers, 22 neurons for the labels, 18 auxiliary neurons for the 9 INC instructions, 18 auxiliary neurons for the 9 DEC instructions, 2 auxiliary neurons for the simulation of two consecutive INC instructions, $3 \cdot 2 = 6$ auxiliary neurons for the two simulations of consecutive $INC - DEC$ instructions, 7 neurons for the INPUT module, and finally 2 neurons for the OUTPUT

module. Considering all these neurons, the system contains a total number $R = 117$ of rules.

For such a system it is uncomfortable to make a detailed analysis of the encoded string as we did for register machines, and thus we will just determine its length. Let us first note that in such a system we have $D = 1$ and $C = 3$, and thus 1 and 2 bits will suffice to represent any delay and any number of consumed spikes, respectively. The 9 neurons that correspond to the registers contain two firing rules each, and thus require 15 bits each, for a total of 135 bits. The 22 neurons associated with the labels contain each one firing and one forgetting rule, for a total of 11 bits that, multiplied by 22, makes 242 bits. Each auxiliary neuron involved in the simulation of the 9 INC instructions contains one firing rule, and thus requires 8 bits to be described; all the 18 neurons require 144 bits. The same argument applies to the 18 auxiliary neurons involved in the simulation of the 9 DEC instructions, thus adding further 144 bits. The two auxiliary neurons used to simulate two consecutive INC instructions also contain one firing rule each, thus contributing with 16 bits. The same applies to the 6 auxiliary neurons used to simulate (two instances of) an INC followed by a DEC, thus adding 48 bits, as well as to the 7 neurons that are used in the INPUT module (56 bits) and the 2 auxiliary neurons of the OUTPUT module (16 bits).

All considered, we need 801 bits to describe the rules contained in the neurons. To these we must add the $m^2 - m = 6972$ bits needed to describe the structure of the synapse graph. We thus obtain a total of 7773 bits to encode the universal standard SN P system presented in [18]. This quantity is an upper bound to $ds_{\mathcal{C}}(\Pi)$, the description size of the system under the proposed encoding, which in turn is an upper bound to $ds(\mathrm{SNP})$, the description complexity of the class of SN P systems. Tighter bounds can be obtained by explicitly computing the encoding of Π and then compressing it by means of a lossless compressor.

By assuming $ds(\mathrm{DRM}) = 263$ and $ds(\mathrm{SNP}) = 7773$, an approximated value of the *redundancy* of spiking neural P systems with respect to deterministic register machines, is:

$$R_{\mathrm{DRM}}(\mathrm{SNP}) = \frac{ds(\mathrm{SNP})}{ds(\mathrm{DRM})} = \frac{7773}{263} \approx 29.56$$

On the other hand, by assuming $ds(\mathrm{DRM}) = 258$ (that results from the computation of the entropy of the string that encodes the universal deterministic register machine depicted in Figure 3.1) the redundancy becomes $R_{\mathrm{DRM}}(\mathrm{SNP}) = \frac{ds(\mathrm{SNP})}{ds(\mathrm{DRM})} = \frac{7773}{258} \approx 30.13$. These results suggest that the description of a universal SN P system is at least 29 or 30 times more verbose with respect to the description of a universal deterministic register machine. This number can certainly be lowered by computing $ds(\mathrm{SNP})$ from the entropy of the bit string used to encode Π. We are currently performing these computations; the results will be exposed during the workshop.

In [18] it is also shown that by allowing firing rules of the extended type it is possible to build a universal SN P system by using only 49 neurons. However this time many neurons have 7 rules instead of 2, and to describe every extended rule $E/a^c \to a^p; d$ we need also some bits to specify the value of p, that does not occur in standard rules. As a result, there may be some doubts about what, among the two systems, is smaller. To find out the winner of this competition, let us compute the description size of the extended SN P system. As reported in [18], this time the system is able to simulate the universal register machine which is composed of $n = 8$ registers and $m = 23$ instructions. The rules contain 12 different regular expressions, do not contain delays, and the maximum number of spikes produced or consumed is 13. Thus we will need 4 bits to specify a regular expression, 0 bits to represent the delays, and 4 bits to represent each number of produced/consumed spikes. The 49 neurons are used as follows: 8 neurons for the registers, 22 neurons for the labels, 13 for the DEC instructions, 5 for the INPUT module, and 1 for the OUTPUT module. Each extended firing rule requires $1 + 4 + 4 + 4 = 13$ bits to be encoded, whereas a forgetting rule requires $1 + 4 = 5$ bits. Recall that each rule is preceded by a 1 in a list of rules, while the list itself is terminated with a 0. Each of the 8 neurons used for the registers contains 2 firing rules ($2 \cdot 13 + 3 = 29$ bits), for a total of 232 bits. Each of the 22 neurons used for the labels contains 3 firing rules and 4 forgetting rules (67 bits), for a total of 1474 bits. Each of the 13 neurons which are used in the simulation of the DEC instructions contains 1 firing rule (15 bits), for a total of 195 bits. Each of the 5 neurons used in the INPUT module also contains 1 firing rule (total: 75 bits), whereas the neuron used in the OUTPUT module contains 2 firing rules and 1 forgetting rule (35 bits). To all these bits we must add the $49^2 - 49 = 2352$ bits which are needed to encode the synapse graph. All considered, we obtain 4361 bits, which is well less than the 7773 bits obtained with the first universal SN P system. Hence this is a tighter upper bound to $ds(\mathrm{SNP})$ and, assuming $ds(\mathrm{DRM}) = 263$ or $ds(\mathrm{DRM}) = 258$, we obtain

$$R_{\mathrm{DRM}}(\mathrm{SNP}) = \frac{4361}{263} \approx 16.58 \qquad \text{and} \qquad R_{\mathrm{DRM}}(\mathrm{SNP}) = \frac{4361}{258} \approx 16.90$$

respectively. Also in this case, tighter bounds can be obtained by explicitly computing the bit string that encodes the universal extended SN P system and then compressing it using a lossless compressor. We are currently performing these computations; the results will be exposed during the workshop.

3.6 UREM P systems Let UREM denote the class of UREM P systems. As proved in [6], in order to reach computational completeness we can restrict our attention to *deterministic* systems in which the skin membrane contains one elementary membrane for each register of the (possibly universal) simulated deterministic register machine. This means getting rid of the membrane structure, saving a lot of bits while describing the system. Similarly, we can restrict our attention to UREM P systems in which the amounts Δe of energy that occur in each rule are taken from the set $\{-1, 0, 1\}$. This means that for each rule 2 bits will suffice to encode the actual value of Δe.

Under these assumptions, in order to encode a particular system $\Pi \in \mathrm{UREM}$ we have to specify as parameters the number n of elementary membranes contained into the skin, the number m of symbols contained into the alphabet, and the number R of rules that occur in the system. Then, for each membrane we have to specify the list of its rules. Just like it happens with SN P systems, in general every membrane will have a different number of rules, and thus we will prepend the description of each rule by a bit equal to 1, and we will conclude each list of rules with a 0. To encode each rule $(op_i : a, \Delta e, b)$ we need 1 bit to specify whether $op = in$ or $op = out$, $2 \cdot \lg m$ bits to specify the alphabet symbols a and b, and 2 bits to express the value of Δe.

$$
\begin{aligned}
0 &: (in_1 : p_0, 0, \widetilde{p}_0),\ (out_1 : \widetilde{p}_0, -1, p_1),\ (out_1 : \widetilde{p}_0, 0, p_2)\\
1 &: (in_7 : p_1, 1, \widetilde{p}_1),\ (out_7 : \widetilde{p}_1, 0, p_0)\\
2 &: (in_6 : p_2, 1, \widetilde{p}_2),\ (out_6 : \widetilde{p}_2, 0, p_3)\\
3 &: (in_5 : p_3, 0, \widetilde{p}_3),\ (out_5 : \widetilde{p}_3, -1, p_2),\ (out_5 : \widetilde{p}_3, 0, p_4)\\
4 &: (in_6 : p_4, 0, \widetilde{p}_4),\ (out_6 : \widetilde{p}_4, -1, p_5),\ (out_6 : \widetilde{p}_4, 0, p_3)\\
5 &: (in_5 : p_5, 1, \widetilde{p}_5),\ (out_5 : \widetilde{p}_5, 0, p_6)\\
6 &: (in_7 : p_6, 0, \widetilde{p}_6),\ (out_7 : \widetilde{p}_6, -1, p_7),\ (out_7 : \widetilde{p}_6, 0, p_8)\\
7 &: (in_1 : p_7, 1, \widetilde{p}_7),\ (out_1 : \widetilde{p}_7, 0, p_4)\\
8 &: (in_6 : p_8, 0, \widetilde{p}_8),\ (out_6 : \widetilde{p}_8, -1, p_9),\ (out_6 : \widetilde{p}_8, 0, p_0)\\
9 &: (in_6 : p_9, 1, \widetilde{p}_9),\ (out_6 : \widetilde{p}_9, 0, p_{10})\\
10 &: (in_4 : p_{10}, 0, \widetilde{p}_{10}),\ (out_4 : \widetilde{p}_{10}, -1, p_0),\ (out_4 : \widetilde{p}_{10}, 0, p_{10})\\
11 &: (in_5 : p_{11}, 0, \widetilde{p}_{11}),\ (out_5 : \widetilde{p}_{11}, -1, p_{12}),\ (out_5 : \widetilde{p}_{11}), 0, p_{13})\\
12 &: (in_5 : p_{12}, 0, \widetilde{p}_{12}),\ (out_5 : \widetilde{p}_{12}, -1, p_{14}),\ (out_5 : \widetilde{p}_{12}), 0, p_{15})\\
13 &: (in_2 : p_{13}, 0, \widetilde{p}_{13}),\ (out_2 : \widetilde{p}_{13}, -1, p_{18}),\ (out_2 : \widetilde{p}_{13}), 0, p_{19})\\
14 &: (in_5 : p_{14}, 0, \widetilde{p}_{14}),\ (out_5 : \widetilde{p}_{14}, -1, p_{16}),\ (out_5 : \widetilde{p}_{14}), 0, p_{17})\\
15 &: (in_3 : p_{13}, 0, \widetilde{p}_{13}),\ (out_3 : \widetilde{p}_{13}, -1, p_{18}),\ (out_3 : \widetilde{p}_{13}), 0, p_{20})\\
16 &: (in_4 : p_{16}, 1, \widetilde{p}_{16}),\ (out_4 : \widetilde{p}_{16}, 0, p_{11})\\
17 &: (in_2 : p_{17}, 1, \widetilde{p}_{17}),\ (out_2 : \widetilde{p}_{17}, 0, p_{21})\\
18 &: (in_4 : p_{18}, 0, \widetilde{p}_{18}),\ (out_4 : \widetilde{p}_{18}, -1, p_0),\ (out_4 : \widetilde{p}_{18}), 0, p_{22})\\
19 &: (in_0 : p_{19}, 0, \widetilde{p}_{19}),\ (out_0 : \widetilde{p}_{19}, -1, p_0),\ (out_3 : \widetilde{p}_{19}), 0, p_{18})\\
20 &: (in_0 : p_{20}, 1, \widetilde{p}_{20}),\ (out_0 : \widetilde{p}_{20}, 0, p_0)\\
21 &: (in_3 : p_{21}, 1, \widetilde{p}_{21}),\ (out_3 : \widetilde{p}_{21}, 0, p_{18})
\end{aligned}
$$

Fig. 3.2 A small universal deterministic UREM P system. In each row, the number on the left refers to the label of the simulated instruction of the register machine depicted in Figure 3.1

A simple encoding of a UREM P system $\Pi(n, m, R)$ containing n elementary membranes into the skin, and R rules that operate on m symbols of the alphabet, is composed of $n + 1$ blocks, one for each membrane. Each block encodes the sequence of rules associated with the corresponding membrane, listing the rules as described above. Each rule requires $4 + 2 \lg m$ bits to be encoded (one bit is used to indicate that we have not yet reached the end of the list of rules), for a total of $2R\,[2 + \lg m]$ bits. One bit is needed to terminate each of the $n + 1$ lists, and thus the description size of the whole system under the encoding $\mathcal{C}$ just proposed is:

$$ds_{\mathcal{C}}(\Pi) = 2R\,[2 + \lg m] + n + 1$$

A small universal UREM P system (here proposed for the first time) can be obtained by simulating the small universal deterministic register machine described in section 3.1. Such a small UREM P system contains $n = 8$ elementary membranes. As recalled in section 2.3 (see [6] for details) we need the objects p_j and $\widetilde{p}_j$, for all $j \in \{0, 1, \ldots, 21\}$, to simulate the instructions of the register machine, as well as the object p_{22} to simulate the jump to the non-existent instruction number 22 (to halt the computation), for a total of $m = 45$ alphabet symbols. Each INC instruction of the register machine requires 2 rules to be simulated, whereas each DEC instruction requires 3 rules, for a total of $R = 57$ rules, depicted in Figure 3.2.

The skin membrane does not contain any rule, and thus the first block of the encoding of Π is 0. The elementary membrane that simulates register 0 contains the five rules that correspond to the INC (line 19 in Figure 3.2) and to the DEC (line 20) instructions that affect the contents of register 0. Since $n = 8$, we will need 3 bits to encode a register number; similarly, to encode an alphabet symbol we will need $\lg m = \lg 45 = 6$ bits. The bit string that encodes membrane 0 is thus:

$$1\underbrace{0}_{in}\underbrace{010011}_{p_{19}}\underbrace{00}_{0}\underbrace{110011}_{\widetilde{p}_{19}}1\underbrace{1}_{out}\underbrace{110011}_{\widetilde{p}_{19}}\underbrace{11}_{-1}\underbrace{000000}_{p_0}$$
$$1\underbrace{1}_{out}\underbrace{110011}_{p_{19}}\underbrace{00}_{0}\underbrace{010010}_{p_{18}}1\underbrace{0}_{in}\underbrace{010100}_{p_{20}}\underbrace{01}_{1}\underbrace{110100}_{\widetilde{p}_{20}}$$
$$1\underbrace{1}_{out}\underbrace{110100}_{\widetilde{p}_{20}}\underbrace{00}_{0}\underbrace{000000}_{p_0}0$$

where we have encoded in as 0, out as 1, $\Delta e = 0$ as 00, $\Delta e = 1$ as 01, $\Delta e = -1$ as 11, p_k as the 5-bit binary encoding of $k \in \{0, 1, \ldots, 22\}$ with an additional leading 0, and $\widetilde{p}_k$ as the 5-bit binary encoding of $k \in \{0, 1, \ldots, 21\}$ with an additional leading 1. Operating in a similar way for all the elementary membranes of Π, we obtain the following binary string (the spaces denote a separation between consecutive rules; they are put here to help the reader, but are not necessary to decode the string):

(Skin membrane)

0

(Membrane 0)

1 001001100110011 1 111001111000000 1 111001100010010
1 001010001110100 1 111010000000000 0

(Membrane 1)

1 000000000100000 1 110000011000001 1 110000000000010
1 000011101100111 1 110011100000100 0

(Membrane 2)

1 000110100101101 1 110110111010010 1 110110100010011
1 001000101110001 1 111000100010101 0

(Membrane 3)

1 000110100101101 1 110110111010010 1 110110100010100
1 001010101110101 1 111010100010010 0

(Membrane 4)

1 000101000101010 1 110101011000000 1 110101000001011
1 101000001110000 1 111000000001011 1 001001000110010
1 111001011000000 1 111001000010110 0

(Membrane 5)

1 000001100100011 1 110001111000010 1 110001100000100
1 000010101100101 1 110010100000110 1 000101100101011
1 110101111001100 1 110101100001101 1 000110000101100
1 110110011001110 1 110110000001111 1 000111000101110
1 110111011010000 1 110111000010001 0

(Membrane 6)

1 000001001100010 1 110001000000011 1 000010000100100
1 110010011000101 1 110010000000011 1 000100000101000
1 110100011001001 1 110100000000000 1 000100101101001
1 110100100001010 0

(Membrane 7)

1 000000101100001 1 110000100000000 1 000011000100110
1 110011011000111 1 110011000001000 0

We can thus conclude that $ds_{\mathcal{C}}(\Pi) = 921$, where $\mathcal{C}$ denotes our encoding.

Also in the case of UREM P systems we can ask how many bits we would save by compressing the above bit string. If we look at such a string we can see that it contains 516 zeros and 405 ones. Hence the probability that a 0 occurs in any given position is $p_0 = \frac{516}{921}$, whereas the probability that a 1 occurs is $p_1 = \frac{405}{921}$. The entropy of the above sequence is thus $H(\Pi) = -p_0 \log_2 p_0 - p_1 \log_2 p_1 \approx 0.990$, and we can conclude that a compressed string produced by an optimal entropy-based compressor, whose length is approximately equal to the length of the uncompressed string times the entropy, would contain $\lceil 911.79 \rceil = 912$ bits. Such a quantity is an upper bound to $ds(\mathrm{UREM})$, the theoretical description size complexity of the class of UREM P systems.

By assuming $ds(\mathrm{DRM}) = 263$ and $ds(\mathrm{UREM}) = 921$, and approximated value of the *redundancy* of UREM P systems with respect to deterministic register machines is:

$$R_{\mathrm{DRM}}(\mathrm{UREM}) = \frac{ds(\mathrm{UREM})}{ds(\mathrm{DRM})} = \frac{921}{263} \approx 3.50$$

On the other hand, by assuming $ds(\mathrm{DRM}) = 258$ and $ds(\mathrm{UREM}) = 912$ (that result by considering the entropies of the corresponding encoded strings) the redundancy becomes $R_{\mathrm{DRM}}(\mathrm{UREM}) = \frac{912}{258} \approx 3.53$. These results suggest that the description of a universal UREM P system is at least 3.5 times more verbose with respect to the description of a universal deterministic register machine.

4 Conclusions

Trying to find a common measure for the size of different computation devices, we have introduced the *description size* of a device M as the length of the binary string produced by a "reasonable" encoding of M. For three classes of computation devices (deterministic register machines, SN P systems and UREM P systems) we have computed the description size of randomly chosen devices, as well as of a universal device taken from each class. In this way we have observed that the smallest universal SN P system currently known has a description which is about 16.58 times as verbose as the the description of the smallest deterministic register machine (currently known), while the smallest universal deterministic UREM P system (here described for the first time) is only about 3.5 times more verbose with respect to the register machine. Indeed, an interesting question that naturally arises after these calculations is: What is the minimum theoretical description size that can be obtained by considering *all* possible universal computing devices? In other words: What is the minimum number of bits which are necessary to describe the structure of a universal computing device?

Making a bit of self-criticism, we are aware of some weak points in our results. First of all, since we are speaking about Turing computable functions we should also consider the many small universal Turing machines which have been defined in the literature. Such a work is currently in progress, as well as the comparison with the description size complexities of small universal tissue and antiport P systems. Moreover, we should

not give for free that the smallest universal computation devices defined in the literature automatically produce the smallest possible description sizes: for example, minimizing the number of neurons in SN P systems does not automatically produce the shortest among all possible encoded bit strings, since the encoding takes into account all the parameters (number of rules, number of consumed spikes, etc.) that may affect the size of the system. Finally, smarter encoding functions could produce shorter strings than those we have presented in this paper, thus showing tighter bounds to the theoretical values of description size complexities. Improving our results under all these points of view is a direction of research of a clear interest. Speaking about the encoding functions, we note that on one hand we required that the encoding and decoding processes should be as simple as possible: their description should require only a finite number of bits. On the other hand, we did not add (nor specify better) such bits to the description size of our computing devices. Whether this is correct or not is questionable, but let us note that also representing the decoding algorithm as a string of bits would require a decoding process, and so on in an endless sequence of encodings and decodings.

Acknowledgments. The work of the authors was supported by MiUR under the project "Azioni Integrate Italia-Spagna - Theory and Practice of Membrane Computing" (Acción Integrada Hispano-Italiana HI 2005-0194), and by project "Models of natural computing and applications" — FIAR 2007 — University Milano–Bicocca.

Bibliography

[1] J.L. Balcázar, J. Díaz, J. Gabarró. *Structural Complexity*. Voll. I and II, Springer-Verlag, Berlin, 1988–1990.

[2] H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez. Spiking Neural P Systems with Extended Rules. In M.A. Gutiérrez–Naranjo, Gh. Păun, A. Riscos–Núñez, F.J. Romero–Campero (eds.), *Fourth Brainstorming Week on Membrane Computing*, RGCN Report 02/2006, Sevilla University, Fénix Editora, Vol. I, 241–265.

[3] A. Church: An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics* 58:354–363, 1936.

[4] M. Cook: Universality in Elementary Cellular Automata. *Complex Systems*, 15:1–40, 2004.

[5] E. Csuhaj–Varjú, M. Margenstern, G. Vaszil, S. Verlan: On Small Universal Antiport P Systems. *Theoretical Computer Science*, 372(2–3):152–164, 2007.

[6] R. Freund, A. Leporati, M. Oswald, C. Zandron: Sequential P Systems with Unit Rules and Energy Assigned to Membranes. In *Proceedings of Machines, Computations and Universality, MCU 2004*, LNCS 3354, Springer-Verlag, Berlin, 2005, pp. 200–210.

[7] R. Freund, M. Oswald: GP Systems with Forbidding Context. *Fundamenta Informaticae*, 49(1–3):81–102, 2002.

[8] R. Freund, M. Oswald: Small Universal Antiport P Systems and Universal Multiset Grammars. In M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Nnez, F.J. Romero-

Campero (eds.), *Proceedings of the* 4^{th} *Brainstorming Week on Membrane Computing*, Sevilla, January 30–February 3, 2006, vol. II, RGNC Report 03/2006, Fénix Editora, Sevilla, 2006, pp. 51–64.

[9] R. Freund, Gh. Păun: On the Number of Non-terminals in Graph-controlled, Programmed, and Matrix Grammars. In M. Margenstern, Y. Rogozhin (eds.), *Proc. Conf. Universal Machines and Computations*, LNCS 2055, Springer–Verlag, Berlin, 2001, pp. 214–225.

[10] R. Freund, Gh. Păun: From Regulated Rewriting to Computing with Membranes: Collapsing Hierarchies. *Theoretical Computer Science*, 312:143–188, 2004.

[11] M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71(2-3):279–308, 2006.

[12] M. Ionescu, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Computing with spiking neural P systems: traces and small universal systems. In C. Mao, T. Yokomori (eds.), *DNA Computing*, 12^{th} *International Meeting on DNA Computing, DNA12*, Revised Selected Papers, LNCS 4287, Springer-Verlag, Berlin, 2006, pp. 1–16.

[13] S. Kleene: A Theory of Positive Integers in Formal Logic. *American Journal of Mathematics*, 57, 1935, pp. 153-173 and 219-244.

[14] I. Korec: Small Universal Register Machines. *Theoretical Computer Science*, 168: 267–301, 1996.

[15] M. Kudlek: Small Deterministic Turing Machines. *Theoretical Computer Science*, 168:241–255, 1996.

[16] M.L. Minsky: Size and Structure of Universal Turing Machines Using Tag Systems. In *Recursive Function Theory, Symp. in Pure Mathematics*, 5:229–238, Amer. Mathematical Soc., Provelence, RS, 1962.

[17] M.L. Minsky: *Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, 1967.

[18] A. Păun, Gh. Păun: Small Universal Spiking Neural P Systems. *BioSystems*, 90:48–60, 2007.

[19] Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143, 2000. See also Turku Centre for Computer Science — TUCS Report No. 208, 1998.

[20] Gh. Păun: *Membrane Computing. An Introduction*. Springer–Verlag, Berlin, 2002.

[21] E. Post: Formal Reductions of the Combinatorial Decision Problem. *American Journal of Mathematics*, 65(2):197–215, 1943.

[22] Y. Rogozhin: Seven Universal Turing Machines. *Math. Issled*, 69:76–90, 1982.

[23] Y. Rogozhin: A Universal Turing Machine with 10 States and 3 Symbols. *Izv. Akad. Nauk. Respub. Moldova Mat.*, 4:80–82 and 95, 1992.

[24] Y. Rogozhin: Small Universal Turing Machines. *Theoretical Computer Science*, 168:215–240, 1996.

[25] Y. Rogozhin, S. Verlan: On the Rule Complexity of Universal Tissue P Systems. In R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa (eds.), *Membrane Computing, International Workshop, WMC 6*, Selected and Invited Papers, LNCS 3850, Springer-Verlag, Berlin, 2005, pp. 356–363.

[26] C.E. Shannon: A Universal Turing Machine with Two Internal States. *Automata*

Studies, Ann. Math. Stud., 34:157–165, Princeton University Press, Princeton, NJ, 1956.

[27] S. Wolfram: *A New Kind of Science*, Wolfram Media, Champaign, IL, 2002.

[28] Wolfram Research and S. Wolfram: The Wolfram 2,3 Turing Machine Research Prize. http://www.wolframscience.com/prizes/tm23/

[29] The P systems Web page: http://ppage.psystems.eu/

Enumerating Membrane Structures

Vincenzo Manca

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

A recurrent formula for enumerating membrane structures is given. It is deduced by elementary combinatorial methods, by providing a simplification with respect to a classical formula for the enumeration of trees, which is based on the analytical method of generating functions.

1 Introduction

The computation of the number of different membrane structures constituted by n membranes was considered at early beginning of Membrane Computing [6], in a preliminary draft of [7]. It is a well known combinatorial problem equivalent to the enumeration of unlabeled unordered trees [4]. Therefore, it is related to Catalan numbers and to a lot of combinatorial problems [2] which recently were proved to be investigated even by Greek mathematicians (*e. g.*, Hypparcus' problem and its modern variant known as Schröder's problem [8]).

For the enumeration of (this kind of) trees, no exact analytical formula is available, but a recurrent formula, based on integer partitions, was given in [5], which was deduced by means of generating functions. In the same paper also a complex asymptotic formula is presented.

In this note, we provide a new recurrent formula related to a simple combinatorial analysis of membrane structures.

The set $\mathbb{M}$ of finite membrane structures can be defined by induction. Let us consider finite multisets as an extension of finite sets, denoted by $\{a, a, b, c\}$, where elements can occur more than one time. A membrane *skin wrapping* of a finite multiset S is an operation which provides a structure, denoted by $[S]$, where the elements of S are *wrapped* in a common membrane. We remark that braces denote multiset theoretic collections, while brackets denote membrane (spatial-topological) inclusions. For a mathematical definition of membrane structures, it is useful to distinguish these two notions and to use both of them. Let us start from the multiset $\{[\,]\}$ including only one occurrence of the *elementary membrane* $[\,]$ (wrapping the empty multiset). The set $\mathbb{M}$ of membrane

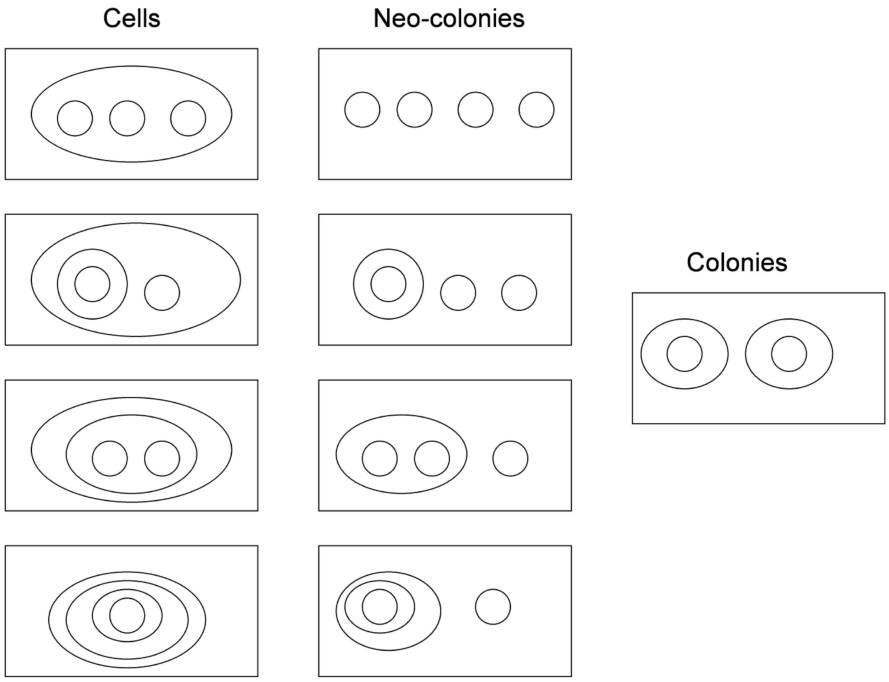


Fig. 1.1 A representation of the membrane structures with four membranes (braces are represented by rectangles and brackets by ovals).

structures is generated by induction by means of the multiset union $+$ (summing the element occurrences of two multisets), which is a binary commutative and associative operation, and by means of the *skin* wrapping operation. The elements occurring in a multiset S of $\mathbb{M}$ are called the *components* of S.

$$\{[\,]\} \in \mathbb{M} \qquad \textit{Base step}$$
$$S,\ S_1,\ S_2 \in \mathbb{M} \implies \{[S]\},\ S_1 + S_2 \in \mathbb{M} \qquad \textit{Inductive step}$$

For example, $\{[\,]\} + \{[\,]\} = \{[\,], [\,]\}$ and $[\{[\,], [\,]\}] = [[\,], [\,]] \in \mathbb{M}$.

2 Cells, Colonies, and Neo-colonies

Given n elements, the number of multisets of k elements over the n elements, according to [1], is given by:

$$\binom{n+k-1}{k} \tag{40}$$

By using formula (40), the following recurrent formula is given in Knuth's book [4] (pag. 386), which provides the number $P(n)$ of membrane structures of type $[S]$ with n membranes ($\mathbb{N}$ is the set of natural numbers, $n > 0$, $P(0) = 1$, $j, n_1, n_2, \ldots, n_j, k_1, k_2, \ldots, k_j \in \mathbb{N}$):

$$P(n) = \sum_{k_1 \cdot n_1 + k_2 \cdot n_2, \ldots k_j \cdot n_j = n} \prod_{i=1,\ldots,\, j} \binom{M(n_i) + k_i - 1}{k_i} \tag{41}$$

Unfortunately, formula (41) is not manageable for an effective computation, because it is based on integer partitions, which grow exponentially (according to Ramanujan's asympthotic formula [3]) making the evaluation of the sum in formula (41) very complex. For this reason, we adopt a different enumeration strategy, by considering the following partition of membrane structures: i) **Cells**, ii) **Neo-colonies**, and iii) **Colonies**. Cells are structure of type $\{[S]\}$ where $S \in \mathbb{M}$, and S is not empty. Neo-colonies are multisets of type $S + \{[\,]\}$, with $S \in \mathbb{M}$, that is, multisets having an elementary membrane as their component. Colonies are structures different from cells and neo-colonies, that is, they are not singleton multisets and do not have components which are elementary membranes. In Fig. 1.1 structures on the left side are cells, structures in the middle are neo-colonies, and the structure on the right is a colony. Cells are easily proved to be equivalent to rooted unlabeled unordered trees, while colonies and neo-colonies represent unlabeled unordered forests. We denote by $M(n)$ the numbers of membrane structures having n membranes (pairs of matching brackets), and by $N(n), P(n), C(n)$, the number of Neo-colonies, Cells, and Colonies, respectively, having n membranes. It easy to realize that a membrane structure with n membranes inside a further external membrane provides a cell with $n + 1$ membranes, while united with the multiset $\{[\,]\}$ provides a neo-colony. The following lemmas are simple consequences of this partition of membrane structures.

Lemma 1 *For $n > 0$ the following equations hold:*

$$M(n) = N(n+1) = P(n+1).$$

Lemma 2 *For $n > 1$*

$$M(n+1) = 2M(n) + C(n+1).$$

Lemma 3 *Let $M_i(n)$ denote the number of colonies having n membranes and exactly i components, then for $n > 1$*

$$C(n) = \sum_{i=2,\lfloor n/2 \rfloor} M_i(n).$$

Proof . At most $\lfloor n/2 \rfloor$ components can be present in a colony with n membranes.

Lemma 4 *For $n > 2$*

$$C(n+1) \leq M(n).$$

Proof . Removing an external membrane, in a component of a colony with $n+1$ membranes, provides a membrane structure with n membranes. Therefore colonies with $n+1$ membranes are at most $M(n)$.

Lemma 5 *For $n > 2$*

$$C(n+1) \geq M(n-1) - 1.$$

Proof . Some colonies with $n+1$ membranes come from the neo-colonies with n membranes, which are $M(n-1)$, by wrapping all their elementary membranes in a unique membrane. This cannot be done in the case of a neo-colony constituted only by elementary membranes. Therefore, $C(n+1) \geq M(n-1) - 1$.

Putting together the two previous lemmas with Lemma 2 we get the following lemma.

Lemma 6 *For $n > 2$*

$$2M(n) \leq M(n+1) \leq 3M(n).$$
$$2^n < M(n+1) < 3^n$$

According to the previous lemmas we see that in the number $M(n+1)$ the part $2M(n)$ refers to cells plus neo-colonies. Therefore, if $M(n)$ is known, the real problem for the computation of $M(n+1)$ is the evaluation of $M(n+1) - 2M(n) = C(n+1)$, that is, the number of colonies with $n+1$ membranes.

In the case of 1, 2, and 3 membranes we have $M(0) = 1$, $M(1) = 1$, $M(2) = 2$, $M(3) = 4$, as it is indicated in the following schema (external braces are not indicated).

1	[ ]
2	[ ], [ ] [[ ]]
3	[ ], [ ], [ ] [[[ ]]] [ ], [[ ]] [[ ], [ ]]

From Lemma 6 we evaluate immediately $M(4) = 2M(3) + 1 = 9$. In fact, $M_2(4) = 1$, because there is only a colony with 4 membranes: [[ ]], [[ ]]. Analogously, $M(5) = 2M(4) + 2 = 18 + 2 = 20$, because there are two colonies with 5 membranes: [[ ]], [[ ], [ ]], and [[ ]], [[[ ]]]. The sequence from $M(1)$ up to $M(12)$ (sequence A000081 of The On-Line Encyclopedia of Integer Sequences [8]) provides the following values:

n	1	2	3	4	5	6	7	8	9	10	11	12
M(n)	1	2	4	9	20	48	115	286	719	1842	4766	12486

Let $\mathbb{N}^*$ be the set of finite sequences over the set $\mathbb{N}$ of natural numbers. If $X \in \mathbb{N}^*$, and $j \in \mathbb{N}$ we denote by $|X|$ the length of X, and by $X(j)$ the number which occurs in X at position j. Let $\Pi_{n,k}$ be the set of partitions of the the integer n as sum of k summands (simple recurrent formulae for generating and counting the cardinality of this set are available). A partition μ of integers is a multiset of integers, let us denote by $\mu(j)$ the number of occurrences of the integer j in μ.

The following operation associates, for any $i \in \mathbb{N}$, a natural number to any sequence $X \in \mathbb{N}^*$.

$$\bigotimes^i X = \sum_{\mu \in \Pi_{|X|+1-i,i}} \prod_{j \in \mu} \binom{X(j) + \mu(j) - 1}{\mu(j)} \tag{42}$$

For $i, j \in \mathbb{N}$, let $M(1, .., j)$ denote the sequence $(M(1), \ldots, M(j))$, then the main lemma follows.

Lemma 7 *For* $n > 2$

$$C(n+1) = \sum_{i=2, \lfloor (n+1)/2 \rfloor} \bigotimes^i M(1, .., n).$$

Proof Outline. Colonies with n+1 membranes may have 2, 3, ..., but at most a number $\lfloor (n+1)/2 \rfloor$ of components. If we fix a number i of components, then i membranes, of the $n+1$ membranes, must be used for the skins of these i components, therefore the remaining $n+1-i$ are partitioned among these components in all the possible ways. In colonies with 2 components $n+1-2$ membranes can be distributed in 2 components. In colonies with 3 components $n+1-3$ membranes can be distributed in 3 components, and so on, up to $n+1-\lfloor (n+1)/2 \rfloor$. In order to compute the number of all possible membrane arrangements, each partition of μ of $n+1$ into i summands must be "read", according to the formula $\prod_{j \in \mu} \binom{M(j)+\mu(j)-1}{\mu(j)}$, on the sequence $M(1,..,n)$ of membrane structure numbers. If a number n_j of membranes is assigned to the component j, with $\sum_{j=1,i} n_j = n+1-i$, and all the n_j summands are different, we easily find $\prod_{j=1,i} M(n_j)$ different membrane structures. However, this simple formula cannot be applied if two, or more, summands are equal. For example, if a partition has three parts, with two equal parts, say $n+1-3 = p+p+q$, then in a corresponding colony of three components q membranes can be arranged in $M(q)$ ways in a component, and p membranes can be arranged in $M(p)$ ways in the other two components. However, in the two components with p membranes the repetitions of the same configurations must be avoided. For this reason, the product $\prod_{j \in \mu} \binom{M(j)+\mu(j)-1}{\mu(j)}$ is used. In fact, when $\mu(j) = 1$, then this formula provides the value $M(j)$, but, when $\mu(j) > 1$, the number of different multisets of $M(j)$ elements with multiplicity $\mu(j)$ is provided. In conclusion, the number of all possible colonies is the sum of $\bigotimes^i M(1,\ldots,n)$ for all possible number i of components.

This lemma suggests an algorithm for computing $C(n+1)$. From Lemmas 2, 3, and 7 the final proposition follows. The application of the formula of Lemma 7, tested for $n = 0,\ldots,11$, provided the same values, previously given, of the sequence A000081.

Proposition 1 *For* $n > 2$

$$M(n+1) = 2M(n) + \sum_{i=2,\lfloor (n+1)/2 \rfloor} \bigotimes^i M(1,..,n).$$

As an example we provide the computation of $C(11)$. According to lemma 7 the value $C(11)$ is given by:

$$C(11) = \sum_{i=2,5} \bigotimes^i M(1,\ldots,10)$$

The integer partitions of 9 in two summands yield the following set:

$$\Pi_{9,2} = \{\{8,1\},\{7,2\},\{6,3\},\{5,4\}\}$$

therefore:

$$\bigotimes^2 M(1,\ldots,10) = \sum_{\mu\in\Pi_{9,2}} \prod_{j\in\mu} \binom{M(j)+\mu(j)-1}{\mu(j)} =$$

$$\left[\binom{286+1-1}{1}\binom{1+1-1}{1}\right]+\left[\binom{115+1-1}{1}\binom{2+1-1}{1}\right]+\left[\binom{48+1-1}{1}\binom{4+1-1}{1}\right]+\left[\binom{20+1-1}{1}\binom{9+1-1}{1}\right] =$$

$$286 + 115\cdot 2 + 48\cdot 4 + 20\cdot 9 = 888.$$

The integer partitions of 8 in three summands yield the following set:

$$\Pi_{8,3} = \{\{6,1,1\},\{5,2,1\},\{4,3,1\},\{4,2,2\},\{3,3,2\}\}$$

therefore:

$$\bigotimes^3 M(1,\ldots,10) = \sum_{\mu\in\Pi_{8,3}} \prod_{j\in\mu} \binom{M(j)+\mu(j)-1}{\mu(j)}$$

$$\left[\binom{48+1-1}{1}\binom{1+2-1}{2}\right] + \left[\binom{20+1-1}{1}\binom{2+1-1}{1}\binom{1+1-1}{1}\right] + \left[\binom{9+1-1}{1}\binom{4+1-1}{1}\binom{1+1-1}{1}\right] +$$
$$\left[\binom{9+1-1}{1}\binom{2+2-1}{2}\right] + \left[\binom{4+2-1}{2}\binom{2+1-1}{1}\right] = 48 + 20\cdot 2 + 9\cdot 4 + 9\cdot 3 + 10\cdot 2 = 171.$$

The integer partitions of 7 in four summands yield the following set:

$$\Pi_{7,4} = \{\{4,1,1,1\},\{3,2,1,1\},\{2,2,2,1\}\}$$

therefore:

$$\bigotimes^4 M(1,\ldots,10) = \sum_{\mu\in\Pi_{7,4}} \prod_{j\in\mu} \binom{M(j)+\mu(j)-1}{\mu(j)} =$$

$$\left[\binom{9+1-1}{1}\binom{1+3-1}{3}\right] + \left[\binom{4+1-1}{1}\binom{2+1-1}{1}\binom{1+2-1}{2}\right] + \left[\binom{2+3-1}{3}\binom{1+1-1}{1}\right] =$$

$9 + 4 \cdot 2 + 4 = 21.$

The integer partitions of 6 in five summands yield the following set:

$$\Pi_{6,5} = \{\{2, 1, 1, 1, 1\}\}$$

therefore:

$$\bigotimes^5 M(1, \ldots, 10) = \sum_{\mu \in \Pi_{6,5}} \prod_{j \in \mu} \binom{M(j) + \mu(j) - 1}{\mu(j)} =$$

$\left[\binom{2+1-1}{1}\binom{1+4-1}{4}\right] = 2.$

In conclusion, $C(11) = 888 + 171 + 21 + 2 = 1082$, therefore:

$$M(11) = 2M(10) + 1082 = 4766.$$

Bibliography

[1] M. Aigner. Discrete Mathematics. *American Mathematical Society*, 2007.

[2] A. Cayley. On the analytical forms called trees, with application to the theory of chemical combinations.*Mathematical Papers, Vol. 9, 427-460, 1875*.

[3] J H.. Conway and R. K. Guy. The Book of Numbers. *Springer-Verlag*, 1996.

[4] D. Knuth. The Art of Computer Programming, Vol. 1, Fundamental Algorithms. *Addison Wesley*,1968.

[5] R. Otter. The Number of Trees. *The Annals of Mathematics*, 2nd Ser. Vol. 49, N. 3, 583-599,1948.

[6] Gh. Păun. Personal Communication, October 1998.

[7] G. Păun. Computing with membranes. *J. Comput. System Sci.*, 61(1): 108–143, 2000.

[8] N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. *Notices of The American Mathematical Society* Vol. 59, N. 8, 912-915, 2003.

Toward an MP model of Non Photochemical Quenching

Vincenzo Manca[1], Roberto Pagliarini[1], Simone Zorzan[2]

[1]Verona University, Computer Science Department,
Strada Le Grazie 15, 37134 Verona, Italy
{vincenzo.manca, roberto.pagliarini}@univr.it

[2]Verona University, Biotechnological Department,
Strada Le Grazie 15, 37134 Verona, Italy
zorzan@sci.univr.it

In this paper we apply the formalism of Metabolic P Systems for modelling an important phenomenon of photochemical organisms, which determines the plant accommodation to the environmental light. By using some experimental data of this phenomenon, we determine an MP system which discovers, in a specific simplified case, the regulation mechanism underling the Non Photochemical Quenching phenomenon and reproduces, with a good approximation, the observed behaviour of the natural system.

1 Introduction

Photosynthetic organisms have a controversial relationship with the light. They need to maximize the amount of absorbed light but avoid the damages that follow from an excess of excitation energy. The excess of light is, in fact, the main cause for the formation of *reactive oxygen species* (ROS), chemical species that are able to oxidize many constitutive molecules of photosynthetic organisms, producing an effect called *photooxidative damage*. In the most extreme cases the damage suffered from organisms can produce macroscopic effects like the lost of characteristic green color (due to the degradation of chlorophyll molecules) or even the death. The phenomenon that helps to deal with quick light excess, and prevent ROS formation, is called *Non Photochemical Quenching*, shortly *NPQ phenomenon/process.* In this phenomenon the excess of light can, in fact, be dissipated through non chemical ways, when the excitation is transmitted to particular molecules that can pass to their unexcited state by emitting heat.

In this work we present a computational model for NPQ phenomenon obtained by the *Metabolic Log-Gain Principle* combined with techniques of multiple regression. This principle was introduced in [15] and developed in [14] for the construction of *Metabolic P models* from experimental data of a given process. *Metabolic P Systems*,

shortly *MP systems*, are a special class of P systems, developed for expressing biological metabolism and signaling transduction. MP systems were introduced in [21] for a better understanding of quantitative aspect of biological systems, meanwhile avoiding the use of complex systems of differential equations. Differently from the classical P systems [7, 25, 26], based on nondeterministic evolution strategies, MP systems have a discrete evolution computed by deterministic algorithms called *metabolic algorithms*, based on a new perspective introduced in [21] and then developed in the following papers [5, 6, 16, 18, 20, 17, 19, 10]. This new perspective can be synthesized by a new principle which replaces the *mass action principle* of Ordinary Differential Equations (ODE), called *mass partition principle*, which defines the transformation rate of object populations rather than single objects, according to a suitable generalization of chemical laws. In [10] it has been demonstrated that exist, under suitable hypotheses and with some approximation, an equivalence between ODE systems and MP models.

Starting from ODE models some significant biochemical processes effectively modeled by MP systems are: Belousov-Zhabotinsky reaction in Brusellator formulation [5, 6], the Lotka-Volterra dynamics [5, 21], the Suitable-Infect-Recovered epidemic [5], the Protein Kinase C activation [6], the circadian rhythms [9] and the mitotic cycles in early amphibian embryos [20]. The MP model of NPQ phenomenon, here developed, is the first MP model completely deduced by using experimental data of observed behaviors and without any use of previous ODE models.

2 NPQ phenomenon: a biological description

In this section a synthetic description of photosynthetic processes that underlies NPQ phenomenon will be given. For a deeper description some reviews on the subject are available in [23, 24].

Light energy is absorbed by plants mainly by means of protein complexes called *light harvesting complexes* (LHC) or *antennae*, which bind many chlorophyll molecules (Chl) that are excited by light radiation. LHC are connected to the *photosystems*, protein super-complexes that host structures called *reaction centers* where the first phases of photosynthetic process occur.

When a chlorophyll molecule absorbs a photon it passes to the *excited state*. Excited states can be transferred to the reaction centers where a chemical reaction called *charge separation* takes place. Here the oxidation of two water molecules produces a stoichiometric amount of electrons, oxygen and hydrogen ions. Electrons are then carried to enzymes which synthesize high energy molecules ATP and NADPH involved in the so-called *dark phase* of photosynthesis (fixation of atmospheric CO_2 to carbohydrates [3], Figure 2.1, arrow 1). Moreover, excited chlorophyll molecules can be deexcited by passing energy to molecules that emit heat resulting from the non photochemical quenching phenomenon (Figure 2.1, arrow 2), they can emit fluorescence radiation (Figure 2.1,

arrow 3), or can decade in the *triplet state*, that can be transferred to oxygen atoms thus generating ROS (Figure 2.1, arrow 9).

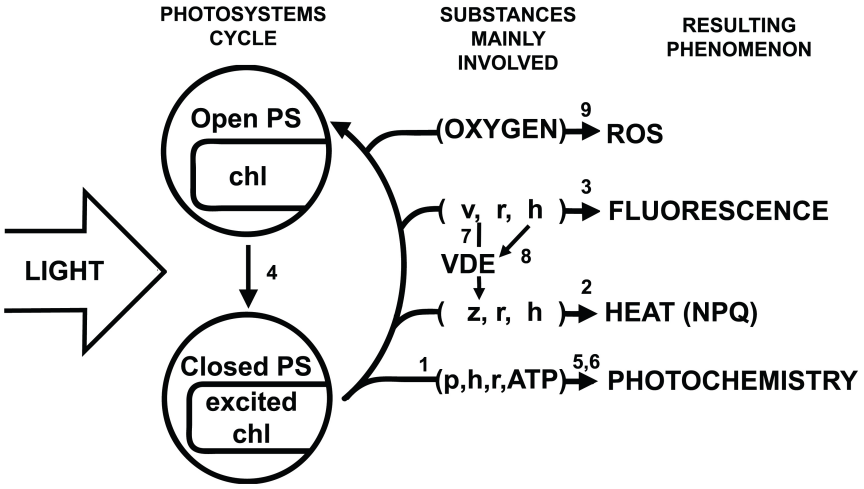


Fig. 2.1 Main actors and relationships of NPQ phenomenon of excitation/deexcitation chlorophylls diagram, according to the following abbreviations: chl = chlorophyll, ps = photosytem, h = hydrogen ion, r = reactivity, p = NADPH, VDE = violaxanthin de-epoxidase, v = violaxanthin, z = zeaxanthin.

In this model photosystem supercomplexes include both reaction centers and antennas, and can be in *closed state* (when photons have been already accepted) or *open* (when the system is able to accept photons).

The LHC in addition to numerous chlorophyll molecules, bind other molecules, called *caroteoinds*, which can absorb energy from excited chlorophyll molecules, to dissipate it by heat generation. Two caroteoinds of great interest in the NPQ phenomenon are *violaxanthin* and *zeaxanthin*. When the absorption of solar radiation exceeds the capacity of the organism to use it, an increase of hydrogen ions provides a signal of overexcitation of the photosystems that triggers a regulative feed-back process. The *violaxanthin de-epoxidase*, shortly VDE, once activated by hydrogen ions, catalyzes the *cycle of xanthophylls*, which transform violaxanthin to zeaxanthin. The presence of zeaxanthin, bound to LCH, favours the NPQ fluorescence and heat production [1], that is respectively, the arrows number 2 and 3 of Figure 2.1.

3 Metabolic P Systems

In an MP system the transition to the next state is calculated according to a *mass partition strategy*, that is, the available substance is partitioned among all reactions which need to consume it. The policy of matter partition is regulated at each instant by *flux*

regulations maps or *flux maps* associated with reactions. The notion of MP system we use comes essentially from [14], where $\mathbb{N}$ and $\mathbb{R}$ respectively denote the sets of natural and real numbers.

Definition 1. (MP system) *An MP system M is a discrete dynamical system specified by the following construct:*

$$M = (X, R, V, Q, \Phi, \nu, \mu, \tau, \sigma_0, \delta)$$

where X, R and V are finite disjoint sets, and moreover the following conditions hold, with $n, m, k \in \mathbb{N}$:

- $X = \{x_1, x_2, \ldots, x_n\}$ *is a finite set of substances (the types of molecules);*
- $R = \{r_1, r_2, \ldots, r_m\}$ *is a finite set of reactions. A reaction is a pair of type $\alpha \to \beta$, with α, β strings over the alphabet X;*
- $V = \{v_1, v_2, \ldots, v_k\}$, $k \in \mathbb{N}$, *is a finite set of parameters provided by a set $H = \{h_v | v \in V\}$ of parameters evolution functions. The function $h_v : \mathbb{N} \to \mathbb{R}$ states the value of parameter v, and the vector $V[i] = (h_v(i) | v \in V)$ represents the values of parameters at the step i;*
- Q *is a set of states viewed as functions $q : X \cup V \to \mathbb{R}$. If we considering an observation instant i ranging in the set of natural numbers, the state q at the instant i can be identified as a vector*

  $$(X[i], V[i]) = (x_1[i], x_2[i], \ldots, x_n[i], v_1[i], v_2[i], \ldots, v_k[i])$$

  of real numbers, constituted by the values which are assigned, by q, to the elements of $X \cup V$;
- $\Phi = \{\varphi_r | r \in R\}$ *is a set of flux regulation maps, where the function $\varphi_r : Q \to \mathbb{R}$ states the amount (moles) which is consumed or produced for every occurrence of a reactant or product of r. We put $u_r[i] = \varphi_r(X[i], V[i])$, also called the (reaction) flux unit of r at the step i, where $U[i] = (u_r[i] | r \in R)$ is the flux units vector;*
- ν *is a natural number which specifies the number of molecules of a (conventional) mole of M, as population unit of M;*
- μ *is a function which assigns, to each $x \in X$, the mass $\mu(x)$ of a mole of x (with respect with to some measure units);*
- τ *is the temporal interval between two consecutive observation steps;*
- $\sigma_0 \in Q$ *is the initial state, that is, $\sigma_0 = (X[0], V[0])$;*
- $\delta : \mathbb{N} \to Q$ *is the dynamic of the system, where $\delta(i) = (X[i], V[i])$ for every $i \in \mathbb{N}$. The vector $V[i]$ is given by the parameters evolution functions, while the substance evolution is given by the following recurrent equation:*

  $$X[i+1] = A \times U[i] + X[i] \tag{43}$$

  where A is the stoichiometric matrix of R over X of dimension $n \times m$ (we define this matrix below), while $\times$, $+$, are the usual matrix product and vector sum.

Definition 2. (Stoichiometric matrix) *The general form of a reaction is $r : \alpha_r \to \beta_r$, where α_r is a string which identifies the multiset of the reactants (substrates) of r and β_r a string which identifies the multiset of the products of r. The stoichiometric matrix A of a set R of reaction over a set X of substances is $A = (A_{x,r} | x \in X, r \in R)$ with $A_{x,r} = |\beta_r|_x - |\alpha_r|_x$, where $|\alpha_r|_x$ and $|\beta_r|_x$ respectively denote the number of occurrences of x in α_r and β_r.*

4 A computational model for NPQ phenomenon

Now, we pose the following question: "*Given a set of experimental data about the NPQ phenomenon, is it possible to determine an MP system having a dynamic in accordance with the experiment, within an acceptable approximation, but which could also predict the future behaviour of the process?*" We will define an MP model that answers to this question.

To represent the NPQ phenomenon it is necessary to know the values of the quantities involved in the phenomenon, along the time. Literature data, complemented by experimental measurements in the laboratory, on Arabidopsis thaliana wild type plants, made it possible to estimate the fundamental values for a selected group of molecules involved in the dissipation of light energy excess.

It is possible to measure the fraction of closed and opened photosystems. The presence of many closed photosystems induces the incapacity of canalizing further amount of energy through the photochemical way: it is the ideal situation to measure the NPQ ability. To induce such a condition strong light flashes called *saturating flashes* are used. With closed photosystems the reduction of the fluorescence yield (i.e. the efficiency of nonphotochemical quenching) can be measured in function of the time. Our model takes into account the reactivity of the system to reach equilibrium after light absorption. The rate of fixation of CO_2 during the NPQ measure condition gives an index of how reactive is the system, as biochemical activity is strictly connected to the capacity of absorbing light energy. The amount of chlorophyll molecules is related to the volume and the surface of the model [2] and the amount of photosystems. The fluorescence and NPQ value can be deducted in arbitrary units[6] (a.u.) from measurements on the sample [22]. NADPH produced has been estimated with laboratory measures, the pH value was deduced by combining data from literature [8, 13, 27] and applying the rate of change to the estimated pH values during the NPQ measure. VDE state and activity was set in relation to various pH values [11]. The change over time of violaxanthin and zeaxanthin was obtained with lab measurements and on the VDE activity [12].

The NPQ process discussed in Section 2 can be expressed by the set of reactions given in Table 4.1. The first three reactions model the possible fates of excited chlorophylls

[6] Arbitrary units are values of suitable observable magnitudes which are proportional to a given phenomenon. They are especially used for evaluating relative variations of variables.

(arrows 1, 2 and 3 in Figure 2.1), the fourth models the fact that the opened photosystems turn into closed photosystems (arrow 4 in Figure 2.1), the fifth and sixth represent the ways that leads the unused products of reaction r_1 to the dark phase of photosynthesis (arrow labeled with numbers 5 and 6 in Figure 2.1). Finally, the seventh and eighth represent xanthophylls cycle (arrows 7 and 8 in Figure 2.1). *Tuners* of a reaction r_i are quantities which influence, with their variations, the variations of the flux units, as it is indicated in Table 4.1.

Reactions	Tuners
$r_1 : c \to o + 12h + p$	c, h, r, p
$r_2 : c \to c + q^+$	c, l, z, r, h
$r_3 : c \to c + f^+$	c, l, v, r, h
$r_4 : o \to c$	o, l, v/z
$r_5 : h \to \lambda$	h, r
$r_6 : p \to \lambda$	p, r
$r_7 : x + 100v \to x + 100z$	x, v
$r_8 : y + h \to x$	y, h

Table 4.1 NPQ reactions and tuners according to the following abbreviations: c = closed photosytems, o = open photosytems, h = hydrogen ions, r = reactivity, p = NADPH, l = light, q^+ = cumulative heat, f^+ = cumulative fluorescence, x = active VDE, y = inactive VDE, v = violaxanthin, z = zeaxanthin.

We introduce the cumulative value of fluorescence and NPQ, called f^+ and q^+ respectively, as new substances which will be useful for the application of Log-Gain theory to our model. The following equations define f^+ and q^+:

$$f^+[j] = \sum_{i=0}^{j} f[i], \quad q^+[j] = \sum_{i=0}^{j} q[i] \tag{44}$$

where the values $f[i]$ and $q[i]$ represent, respectively, the amount of fluorescence and NPQ observed in the system at the step i.

Almost all elements occurring in the definition of MP system are know, because they are deduced by macroscopic observations of the biological phenomenon under investigation. The only component which can't be directly deduced is the set of flux regulation functions. The problem is that each element of Φ depends on the internal microscopic processes in which molecules are involved [14]. This means that the key point, for defining an MP system modelling the NPQ process, is the determination of the set Φ of functions.

We can understand, from the Definition 1, that the knowledge of the number of moles transformed by any rule in the temporal interval between two consecutive observation

steps are essential for the calculation of biological dynamics by means of MP systems. The first step, in order to approximate the flux regulation maps, is to discover the numeric values of the reaction fluxes at each observation step. In order to determine these values we use the *Log-Gain Principle* for MP systems, introduced in [15], which can be seen as a special case of the fundamental principle called *allometry* [4]. In this principle it is assumed that a proportion exists between the relative variation of u_r and the relative variations of tuners of r (in Table 4.1 the set of tuners, for each reaction, are given). In more formal terms, the relative variation of an element $w \in X \cup V$ is expressed, in differential notation, by $d(\lg w)/dt$, where the term *log-gain* comes from [28]. We use a discrete notion of log-gain, given by the following equation:

$$Lg(w[i]) = (w[i+1] - w[i])/w[i] \tag{45}$$

on which the following principle is based.

Principle 1. (Log-Gain) *Let $U[i]$, for $i \geq 0$, be the vector of fluxes at step i. Then the Log-Gain regulation can be expressed in terms of matrix and vector operations:*

$$(U[i+1] - U[i])/U[i] = B \times L[i] \quad + \quad C \times P[i+1] \tag{46}$$

where:

- $B = (p_{r,w} | r \in R, w \in X \cup V)$ *where* $p_{r,w} \in \{0, 1\}$ *with* $p_{r,w} = 1$ *if* w *is a tuner of* r *and* $p_{r,w} = 0$ *otherwise;*
- $L[i] = (Lg(w[i]) | w \in X \cup V)$ *is the column vector of substances and parameters log-gains ;*
- $P = (p_r | r \in R)$ *is the column offset vector. This vector is constituted by the difference between tuner log-gains and flux log-gains;*
- C *is a column binary vector of* 0 *and* 1 *which selects some offsets for obtaining a univocally solvable square linear system (see [14] for a detailed explanation);*
- $\times, +, -, /$ *are, in this context (with some abuse of notation) the product, the sum, the subtraction and division over matrix.*

We call $LG[i]$ the system of equations obtained by (46).

In an MP system the matrix B and vector $L[i]$ are determined by using the biological information. In NPQ phenomenon the Log-Gain Principle provides the following matrix B and vector $C \times P[i+1]$:

$$\begin{bmatrix} 0\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0 \\ 0\,1\,1\,0\,0\,0\,0\,1\,0\,0\,1\,1\,0\,0\,0 \\ 0\,1\,1\,0\,0\,0\,1\,0\,0\,0\,1\,1\,0\,0\,0 \\ 1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1 \\ 0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0 \end{bmatrix}, \quad \begin{bmatrix} p_1[i+1] \\ p_2[i+1] \\ p_3[i+1] \\ 0 \\ p_4[i+1] \\ p_5[i+1] \\ p_6[i+1] \\ p_7[i+1] \end{bmatrix}$$

The transposed $L[i]^T$ of vector $L[i]$ of equation (46) is specified by the following equation:

$$L[i]^T = Lg([\,a[i]\ c[i]\ h[i]\ p[i]\ x[i]\ y[i]\ v[i]\ z[i]\ f^+[i]\ q^+[i]\ l[i]\ r[i]\ f[i]\ q[i]\ (v/z)[i]\,])$$

We reduce the stoichiometric matrix by removing rows which are linearly dependent on other rows (continuing to call A this reduced matrix) and we obtain the following system of equations, called $SD[i]$ [15], which represents the substances variations

$$X[i+1] - X[i] = A \times U[i] \tag{47}$$

where:

$$A = \begin{bmatrix} -1\,0\,0\,1 & 0 & 0 & 0 & 0 \\ 0\,1\,0\,0 & 0 & 0 & 0 & 0 \\ 0\,0\,1\,0 & 0 & 0 & 0 & 0 \\ 12\,0\,0\,0 & -1 & 0 & 0 & -1 \\ 1\,0\,0\,0 & 0 & -1 & 0 & 0 \\ 0\,0\,0\,0 & 0 & 0 & -100 & 0 \\ 0\,0\,0\,0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad X = \begin{bmatrix} c[i] \\ q^+[i] \\ f^+[i] \\ h[i] \\ p[i] \\ v[i] \\ x[i] \end{bmatrix}$$

If we assume to know the flux unit vector at step i and put together the equations (46) and (47) at steps i and $i+1$ respectively, we get a linear system called *offset log-gain adjustment* module at step i, shortly $OLGA[i]$, in which the number of variables is equal to the number of equations. In [14] it is proved that the $OLGA[i]$ of an MP system M is univocally solvable for any step $i \geq 0$. In this way we obtain the following $OLGA[i]$ system for NPQ process, where variables are in bold font and constitute the flux unit vector $U[i+1]$ and the offset vector $P[i+1]$:

$$\begin{aligned} A \times \mathbf{U}[\mathbf{i+1}] &= X[i+2] - X[i+1] \\ \mathbf{U}[\mathbf{i+1}] - (C \times \mathbf{P}[\mathbf{i+1}]) \times U[i] &= (U[i] \times (B \times L[i])) + U[i] \end{aligned} \tag{48}$$

Now, as the vectors $X[i]$ and $V[i]$, for $1 \leq i \leq 810$, are given by experimental measures, we solve the system (48) for $i = 0, \ldots, 809$ obtaining the vector $U[i]$ for $i \in [1, 810]$. This procedure requires the knowledge of $U[0]$. Actually, there are several possibilities

under investigation to obtain this vector. In our case, we consider a system composed by the set of equations $LG[i] + SD[i]$ and we used a suitable iterative technique [15] in order to solve a non-linear system of equations giving a good approximation of $U[0]$.

Given the values of the unit reactions for a sequence of steps, the problem, as previously stated, is to discover the set Φ of flux regulation functions. Although a flux regulation map φ_r depends on the state of the MP system, we can assume that only some substances and parameters are relevant for it. We call these elements *regulators* of φ_r. We use standard multiple regression techniques to find an approximation of φ_r (with respects to its regulators). The resulting functions, given in Table 4.2, approximate the regulation function Φ associated to each reaction $r \in R$. It is possible to see in Figure 4.2 that the behaviors of fluorescence and heat obtained by our MP model are in accordance with the observed values (they are the most interesting parameters of the phenomenon).

Reactions	Flux regulation maps
$r_1 : c \to o + 12h + p$	$\varphi_{r_1} = \alpha_1 + \beta_1 ol + \gamma_1 cl + \eta_1 rl + \vartheta_1 hlp + \rho_1 \frac{v}{z} l$
$r_2 : c \to c + q^+$	$\varphi_{r_2} = \alpha_2 + \beta_2 c + \gamma_2 r + \eta_2 z + \vartheta_2 l + \rho_2 h$
$r_3 : c \to c + f^+$	$\varphi_{r_3} = \alpha_3 + \beta_3 ch + \gamma_3 v + \eta_3 r^{-1} l$
$r_4 : o \to c$	$\varphi_{r_4} = \alpha_4 + \beta_4 ol + \gamma_4 cl + \eta_4 rl + \vartheta_4 hlp + \rho_4 \frac{v}{z} l$
$r_5 : h \to \lambda$	$\varphi_{r_5} = \alpha_5 + \beta_5 ol + \gamma_5 cl + \eta_5 rl + \vartheta_5 hlp + \rho_5 \frac{v}{z} l$
$r_6 : p \to \lambda$	$\varphi_{r_6} = \alpha_6 + \beta_6 ol + \gamma_6 cl + \eta_6 rl + \vartheta_6 hlp + \rho_6 \frac{v}{z} l$
$r_7 : x + 100v \to x + 100z$	$\varphi_{r_7} = \alpha_7 + \beta_7 v + \gamma_7 x$
$r_8 : y + h \to x$	$\varphi_{r_8} = \alpha_8 + \beta_8 y + \gamma_8 h$

Table 4.2 NPQ reactions and flux regulation maps. The values of polynomial coefficients can be downloaded from [29].

5 Conclusions

Our proposed model of NPQ phenomenon, allowed to reproduce quite accurately experimental results for the Arabidopsis wild type case. The predictive ability of computational experiments are so far limited, but it is our objective to progress to an MP model that will be a valuable tool to suggest biological phenomena easily observable in silicio, like pH values or effects of mutations.

The analysis of NPQ process here reported is oversimplified in many aspects. In fact, two kinds of photosystems are involved which play collateral role and may influence the overall dynamics of chlorophyll deexcitation. It is out of the aim of this paper to take into account these more detailed aspects. However, we limit ourselves to reproduce in our mathematical model the observed data of NPQ phenomenon, where fluorescence and

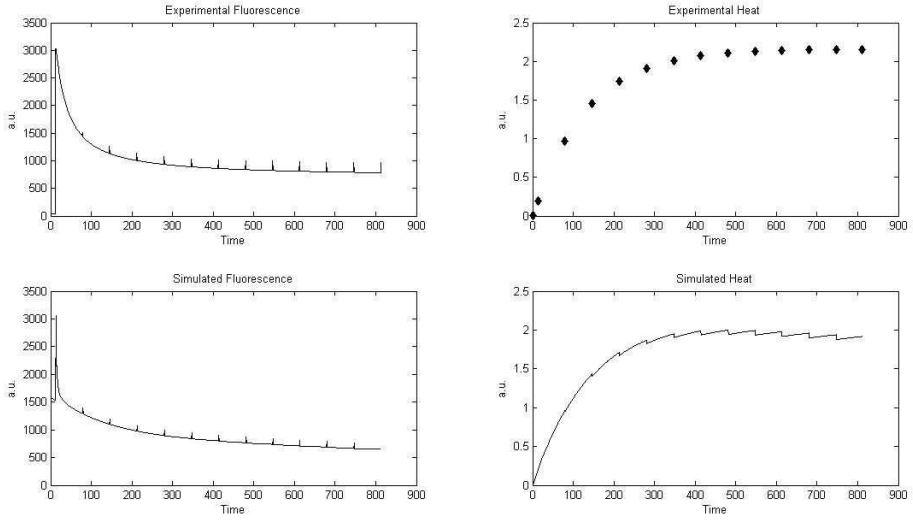


Fig. 4.2 Value of the fluorescence and NPQ, in a. u., during the experimental measurements (top) and simulation results obtained by MP system (bottom). Notice that while experimental heat is measured in correspondence of saturating flash, simulated one is continuous. Our results allow to hypothesize that also the NPQ values are perturbated by flash of light.

heat curves show a particular shape related to the efficiency of this mechanism to dissipating excess of light energy. An interesting property which is predicted by our model is that fluorescence decreases in dependence of photochemical activity (*photochemical quenching*) and non photochemical dissipation. It will be matter of future investigation to take into account other relevant aspects, when reliable data should be available for extending out analysis. In particular, we want to consider some different ways that lead to fluorescence decrease (i.e. LHC migration between photosystems and ROS damages consequences).

The theory of MP systems is evolving and crucial tasks remain to be performed for a complete discovery of the underlying MP dynamics which explains the observed dynamics. The principal investigations are directed to the study of systematic ways for deducing the flux regulation maps from the time series of flux vectors. This problem is directly related to the search of the best regulators associated to reactions. Different research lines are active in this direction and important roles will be given by statistics and genetic programming. However, the modeling of real important biological phenomena is an essential activity for orientating the research of general methods and principles for the theory of MP systems.

Acknowledgments. We are grateful with Prof. Bassi's group of Biotechnological Department, at University of Verona, for laboratory analysis and Petronia Carillo, Department of Life Sciences, Second University of Naples, for CO_2 uptake measurements in relation to the experiment relevant to us.

Bibliography

[1] T.K. Ahn, T.J Avenson, M. Ballottari, Y.C Cheng, K.K. Niyogi, R. Bassi, and G.R. Fleming. Architecture of a charge-transfer state regulating light harvesting in a plant antenna protein. *Science*, 9(320(5877)):794–797, 2008.

[2] N.N. Alder and S.M. Theg. Energetics of protein transport across biological membranes: a study of the thylakoid ΔpH-dependent/cptat pathway. *Cell*, 112:231–242, 2003.

[3] A. Benson and M. Calvin. Carbon dioxide fixation by green plants. *Annual Review of Plant Physiology and Plant Molecular Biology*, 1:25–42, 1950.

[4] L. von Bertalanffy. *General Systems Theory: Foundations, Developments, Applications*. George Braziller Inc, New York, NY, 1967.

[5] L. Bianco, F. Fontana, G. Franco, and V. Manca. P systems for biological dynamics. In *[7]*, pages 81–126. Springer, 2006.

[6] L. Bianco, G. Fontana, and V. Manca. P systems with reaction maps. *International Journal of Foundations of Computer Science*, 17(1):27–48, 2006.

[7] G. Ciobanu, G. Păun, and M.J. Pérez-Jiménez, editors. *Applications of Membrane Computing*. Springer, 2006.

[8] Y. Evron and R.E. McCarty. Simultaneous measurement of deltapH and electron transport in chloroplast thylakoids by 9-aminoacridine fluorescence. *Plant Physiol*, 124:407–414, 2000.

[9] F. Fontana, L. Bianco, and V. Manca. P systems and the modeling of biochemical oscillations. In R. Freud, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, WMC 2005,LNCS*, volume 3850, pages 199–208. Springer, 2005.

[10] F. Fontana and V. Manca. Discrete solution to differential equations by metabolic P systems. *Theoretical Computer Science*, 372:165–182, 2007.

[11] A. Gisselsson, A. Szilagyi, and H. Akerlund. Role of histidines in the binding of violaxanthin de-epoxidase to the thylakoid membrane as studied by site-directed mutagenesis. *Physiol. Plant.*, 122:337–343, 2004.

[12] A.R Holzwarth. Applications of ultrafast laser spectroscopy for the study of biological systems. *Q. Rev. Biophys*, 22:239–295, 1989.

[13] A. Kanazawa and D. M. Kramer. In vivo modulation of nonphotochemical exciton quenching (NPQ) by regulation of the chloroplast atp synthase. *PNAS*, 99(20):12789–12794, 2002.

[14] V. Manca. Log-Gain Principles for Metabolic P Systems. In *G. Rozenberg Festschrift*. To appear.

[15] V. Manca. The Metabolic Algorithm: Principles and Applications. *Theoretical Computer Science*. http://dx.doi.org/10.1016/j.tcs.2008.04.015. In print.

[16] V. Manca. Topics and problems in metabolic P systems. In G. Păun and M.J. Pérez-Jiménez, editors, *Membrane Computing (BWMC4)*. Fenix Editora, Sevilla, Spain, 2006.

[17] V. Manca. Metabolic P Systems for Biochemical Dynamics. *Progress in Natural Science*, 17(4):384–391, 2007.

[18] V. Manca. MP systems approaches to biochemical dynamics: Biological rhythms and oscillation. In H.J. Hoogeboom G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, WMC 2006*, LNCS 4361, 8699. Springer, 2007.

[19] V. Manca. Discrete Simulation of Biochemical Dynamics. In M. H. Garzon and H. Yan, editors, *DNA 13*, LNCS 4848, pages 231–235. Springer, 2008.

[20] V. Manca and L. Bianco. Biological networks in metabolic P systems. *BioSystems*, 91(3):489–498, 2008.

[21] V. Manca, L. Bianco, and F. Fontana. Evolutions and oscillations of P systems: Applications to biological phenomena. In G.Mauri, M.J. Pérez-Jiménez, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, WMC 2004, LNCS*, volume 3365, pages 63–84. Springer, 2005.

[22] K. Maxwell and G.N. Johnson. Chlorophyll fluorescence - a practical guide. *Journal of Experimental Botany*, 51(345):659–668, 2000.

[23] N. Nelson and A. Ben-Shem. The complex architecture of oxygenic photosynthesis. *Nature Reviews Molecular Cell Biology*, 5:971–982, 2006.

[24] N. Nelson and C. Yocum. Structure and Function of Photosystems I and II. *The Annual Review of Plant Biology*, 57:521–565, 2006.

[25] G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.

[26] G. Păun. *Membrane Computing*. Springer, 2002.

[27] B.V. Trubitsin and A.N. Tikhonov. Determination of a transmembrane pH difference in chloroplasts with a spin label tempamine. *Journal of Magnetic Resonance*, 163:257–269, 2003.

[28] E. O. Voit. *Computational Analysis of Biochemical Systems*. Cambridge University Press, 2000.

[29] Web Pages of polynomial coefficients associated to flux regulation maps of NPQ phenomenon: http://profs.sci.univr.it/∼manca/draft/npq-coefficients.pdf.

Applications of Page Ranking in P Systems

Michael Muskulus

Leiden University, Mathematical Institute,
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
muskulus@math.leidenuniv.nl

The page rank of a webpage is a numerical estimate of its importance. In Google's PageRank algorithm the ranking is derived as the invariant probability distribution of a Markov chain random surfer model. The crucial point in this algorithm is the addition of a small probability transition for each pair of states to make the transition matrix irreducible and aperiodic. We show how the same idea can be applied to P systems and allows to define a probability distribution on the objects, resulting in a new complexity measure for P systems. Another interesting application is the pathway identification problem, where information about biochemical reactions from public databases allows to construct a metabolite graph. The invariant distribution should allow to search pathways in this graph more efficiently than the degree weighted algorithms used at the moment. From such automatic pathway calculations one can then construct P system models for targeted metabolic compounds and their reactions.

1 Introduction

Page ranking is the process of assigning a quantitative measure of "authority" to a webpage. Internet search engines usually use a combination of key word related measures and general page ranks to order the results of a user query. These results are displayed in a linear order, and the higher the combined rank of a webpage, the higher in the resulting list it is displayed. Since a higher rank means a higher visibility, there has developed a large commercial interest in optimizing a webpage's content with the goal of improving its ranking, and nowadays the activity of *search engine optimization* has become a full-time job for many people. On the other hand, users of a search engine expect results that lead them to their desired search goals efficiently, so in a way a search engine should optimize their ranking methods with regards to user preferences. In particular, it can be argued that a search engine should use ranking strategies which are objective and unbiased. But note that this leads to a dilemma: if a search engine would openly publish its ranking algorithms, on the one hand this would benefit its users, since then they could, in principle at least, target their queries better. On the other hand, this knowledge would enable owners and designers of webpages to target their desired audience by specific search engine optimization strategies — which might not be what users desire. At the moment, search engines therefore keep their algorithms and ranking methods as closely

guarded secrets. This is, of course, not the only possible solution, but seems to also stem from (i) considerations about competition between distinct search engines, and (ii) probably the assumption that the benefit for the common user would be negligible, since on the average s/he would not be able to understand the algorithms, whereas commercial companies would.

A particular case is Google, probably the most important general purpose search engine of today. It is believed by professional consultants that its page ranking methods take into account more than 200 distinct factors[7], but Google states that the "heart of their software" is an algorithm called *PageRank* [13], whose name seems to be inspired by the last name of Google founder Lawrence Page [34].

The original ranking algorithm behind Google has been published [6, 29] and is also patented (sic!) as a "Method for node ranking in a linked database" (US patent no. 6.285. 999), assigned to Stanford University. It can be shown that *PageRank* is natural in the sense that a few axioms, motivated by the theory of social choice, uniquely characterize *PageRank* [1]. Interestingly, the same method has recently been proposed as a new method of citation analysis that is more authoritative, as self-citations have less impact than in traditional citation analysis [24].

In the following we will describe applications of page ranking in the area of membrane systems. We will specifically concentrate on the original *PageRank* algorithm, since it is closely related to Markov chain modelling of dynamical P systems as in [27]. The applications that we will discuss are (i) defining the *asymptotic behavior* of dynamical P systems, (ii) *approximating* this asymptotic behavior in a manageable way, (iii) defining a new *complexity measure* for dynamical P systems, and (iv) applications in the *identification* of P systems, where biochemical databases are used to suggest interesting P system models via pathway extraction.

2 The PageRank algorithm

The description of the *PageRank* algorithm is usually given in terms of the so-called webgraph. This is the directed graph $D = (V, A)$ where each node $u \in V$ represents a webpage and each arc $(u, v) \in A \subseteq V^2$ represents a link. A link from page $u \in V$ to $v \in V$ can be thought of as providing evidence that v is an "important" page or, more generally, as a *vote* for page v. Intuitively, the more authorative page u itself is, the higher its vote for page v should count, leading to a recursive definition as follows. Let

$$\begin{aligned} r: \quad & V \to \mathbb{R}_+ \\ & v \mapsto r(v) \end{aligned}$$

[7] An analysis of the most important factors used by Google can be found on `http://www.seomoz.org/article/search-ranking-factors`.

be the *ranking function* that assigns a numerical value $r(v)$ to each node v in the webgraph. Then

$$r(v) := \sum_{u \in \{V \mid (u,v) \in A\}} \frac{r(u)}{\mathrm{outdeg}(u)}$$

where the sum runs over all nodes u linking to the page v and $\mathrm{outdeg}(u)$ is the outdegree of node u. In the above interpretation, each page thus transfers its own *PageRank* value equally to all of its link targets. Note that webpages can link multiple times to the same page, but that this counts as only one link, i.e. one arc in the webgraph.

To see that the *PageRank* ranking function is well defined, we need to turn to the theory of Markov chains [5]. Since the webgraph is finite, the function r can be normalized such that $\sum_{v \in V} r(v) = 1$. One can then interprete $r \in \mathbb{R}_+^{|V|}$ as a probability distribution over the set V of webpages. The *transition matrix*

$$P_{uv} = \begin{cases} 1/\mathrm{outdeg}(u) & \text{if } (u,v) \in A, \\ 0 & \text{if } (u,v) \notin A \end{cases}$$

then corresponds to the model for a person surfing between web pages, from now on simply addressed as a *surfer*, described in [6]. In this so-called *random surfer model* a surfer is considered who randomly follows links, without any preference or bias. The matrix P_{uv} then describes the probability for the surfer, being at page u, to visit page v next. The *PageRank* definition is then equivalent to the following matrix equation:

$$r = P^t r.$$

In the language of Markov chain theory this means that r is required to be a *stationary distribution*. In other words, if a large number of random surfers find themselves, at the same time, at webpages distributed according to these probabilities, then after randomly following a link, the individual surfers would end up at different pages, but the number of surfers visiting each webpage would stay approximately the same (exactly the same in the limit of an infinite number of surfers).

Markov chain theory tells us when such a stationary distribution exists and when it is unique. By the ergodic theorem for Markov chains, an *aperiodic* and *irreducible* transition matrix P is sufficient. The transition matrix is aperiodic if the least common multiple of all possible circuits in the webgraph is trivial. This can always be assumed for general digraphs, since only very special digraphs are periodic. Irreducibility is the requirement that each webpage is reachable from each other page, i.e. that the webgraph is strongly connected, and this is usually *not* fulfilled by the transition matrix. In particular, the webgraph usually has pages without outbound links, so-called *dangling* pages or, in the language of Markov chain theory, *sinks* or *black holes*. If one were to apply the *PageRank* idea to a digraph with one or more of these, they would effectively absorb all probability, since eventually a random surfer would always end up in a black hole and stay there forever. To be more precise: One would expect that the resulting invariant

distribution would be zero for all non-sinks, and each sink would be assigned the probability of ending up in it, starting from a random page, in accordance with the random surfer model. However, this is not true. There simply would not exist any stationary distribution in such a case. This "singular" behavior led some people to call such pages black holes, since the usual laws of Markov chain theory cease to work, when one of these is encountered.

The solution to this problem is the truly original idea of the founders of Google: In analogy with the random surfer model, it is assumed that a surfer ending on a sink gets bored and turns *randomly* to a new page from the whole webgraph, which is called *teleportation* in [17]. Of course, this is a somewhat unrealistic model for actual internet user behavior, since how does a surfer *find* a *random* webpage (and with uniform probablity)? But changing the transition matrix accordingly,

$$\bar{P}_{uv} = \begin{cases} 1/\mathrm{outdeg}(u) & \text{if } (u,v) \in A, \\ 1/|V| & \text{if } \mathrm{outdeg}(u) = 0, \\ 0 & \text{if } \mathrm{outdeg}(u) > 0 \text{ and } (u,v) \notin A \end{cases}$$

leads to a matrix with irreducible *blocks* (which is still not irreducible, though, except in special cases). Finally, extending this idea and assuming that the surfer has a certain chance $\alpha > 0$ of turning to a random page *every* time s/he follows a link, leads to

$$\bar{\bar{P}}_{uv} = \begin{cases} 1/|V| & \text{if } \mathrm{outdeg}(u) = 0, \\ \alpha/|V| & \text{if } \mathrm{outdeg}(u) > 0 \text{ and } (u,v) \notin A, \\ \alpha/|V| + (1-\alpha)/\mathrm{outdeg}(u) & \text{if } (u,v) \in A \end{cases} \quad (49)$$

which is truly an irreducible and aperiodic matrix [23]. The stationary distribution r is then, also by the ergodic theorem, an *asymptotic distribution*. This means that a random surfer, starting at an arbitrary webpage, has the chance $r(u)$ to be at page $u \in V$, if he has followed a large number of links, using P_{uv} as transition matrix:

$$\lim_{n\to\infty} (P^t)^n x_0 = r, \quad (50)$$

independent of the initial distribution x_0, i.e. his/her starting page(s). Note that there is a probability $\alpha/|V|$ that the random surfer *stays* at the same page (we can also say that the surfer *accidentally* "jumps" to the same page that he comes from), i.e. we explicitly allow self-transitions here, since it makes the mathematical analysis simpler.

These results are consequences of the Perron-Frobenius theorem [3], which also shows that r is the (normalized) *dominant eigenvector* of P^t, i.e. the corresponding eigenvalue $\lambda = 1$ is the largest eigenvalue P^t possesses. In practice, the direct computation of the dominant eigenvector for the (sparse) transitition matrix of the webgraph is very difficult, due to the graph's enormous size. On the other hand, Eq. 50, starting from the uniform distribution $x_0(u) = 1/|V|$ can be used, and is usually called the *power method* [12]. See [17] for further improvements.

3 Leaky P systems

P system is a general term to describe a broad class of unconvential models of computation that are usually based on multiset rewriting in a hierarchical structure of so-called membranes [31], but also include computational models based on other mechanisms, for example string or grammar rewriting. Originally introduced by Gheorghe Păun in a seminal paper [30], nowadays there exists a large community of researchers working on and with different extensions and variants of P systems.

How are we to interprete the above changes in the context of P systems, i.e. when we are thinking about the random surfer model with possible jumps (Eq. 49) not only as mathematically necessary and convenient, but rather as a real feature of a P system? Obviously, such a mechanism can turn the multisets that describe the object content of a P system into completely different multisets — and we need to control the outcome of such an operation somehow, since otherwise we would end up with an infinite number of possibilities, as multisets are (usually) unbounded.

Let us first consider the case of a probabilistic P system as in [9]. Starting from an initial configuration (multiset) c_0 the evolution of a probabilistic P system generates a rooted tree. The leaves $\mathcal{L}$ of this tree are the halting states and each halting state $h \in \mathcal{L}$ is reached with a distinct probability p_h, where $\sum_{h\in\mathcal{L}} p_h = 1$. If we now introduce additional transitions from each halting state back to the initial state c_0, we have an irreducible Markov chain. This system could be periodic, but it is easy to see that for such a (finite) "closed tree" an invariant probability distribution exists as in the case of an irreducible and aperiodic Markov chain. In fact, the invariant distribution for a state $i \in S$ is given by $\mu_i = 1/|S| \cdot p_{c_0,i}$, where $p_{c_0,i}$ is the probability of reaching the state i when starting from c_0. The factor $1/|S|$ has been introduced such that $\sum_{i\in S} \mu_i = 1$. So we see that the concept of invariant distribution generalizes the probability of reaching a halting state in a probabilistic P system. However, this is not the same as the dynamics proposed in Eq. 49. It is interesting to ask how the invariant measure changes when we additionally introduce the teleportation property exhibited by Eq. 49.

Problem 3.1 *Given a finite Markov chain on a set V with a unique invariant distribution $\mu \in \mathbb{R}_+^{|V|}$, how does μ change when we replace the transition probabilities p_{ij} by $(1-\alpha)p_{ij} + \frac{\alpha}{|V|}$?*

To our knowledge, there has not been much progress on this question, although numerical studies have been done in case of the webgraph (confer [23] for references).

Let us now consider a more general situation. Assume that at each time step[8] there is a

[8] If time is assumed to be continuous, as in dynamical P systems that are simulated by Gillespie-type algorithms, there is still a discrete sequence of events, and by introducing an exponential waiting time distribution for such an event the same comments also apply to this case.

small probability for each object to change spontaneously into another object, analogous to mutations in DNA, where one base can suddenly change into one of the other three possible bases (such a change could be caused by UV radiation, for example). Of course, the objects undergoing such a change cannot be used in another rule at this time step. In fact, by adding rules of the form $u \to v$ for each possible pair of objects (u, v), we can realize this mechanism easily. Let us call a P system with this property a *leaky* P system. Note that leaky P systems are not always irreducible, as the simple example of a system with the rule $2A \to B$ shows: From the state with only one B we can never get to a state with more than one A, although the opposite is possible. However, if all rules were reversible, a leaky P system would be irreducible and have an invariant distribution.

Interesting as these thoughts are, they lead too far here, but we think that leaky P systems should be investigated more closely. For example, a successful computation in a leaky P system would need to be robust against the continuous possibility of small changes of its objects. How could this be realized? Moreover, the following two problems should be looked into:

Problem 3.2 *Given an irreducible aperiodic Markov chain, when adding the teleportation property of Eq. 49 (for some choice of $\alpha > 0$), do the corresponding invariant distributions $\mu(\alpha)$ converge in the limit $\alpha \to 0$?*

Problem 3.3 *Although the random surfer model does not apply to a leaky P system, does the invariant distribution of a leaky P system, in case it exists, converge against the same invariant distribution as in the random surfer model (of the same underlying P system), in the limit that $\alpha \to 0$?*

Intuition suggests that the answers to both questions should be affirmative.

4 Asymptotic behavior of P systems

A particular interesting application area for P systems is the emerging discipline of systems biology [21], and in the past years a number of biological systems have been simulated and anlyzed by P systems [8, 33]. It should be noted, however, that this line of research is only a small part of the total work on P systems, so we consider P systems from a particular perspective here.

The original state-transition P systems are characterized by a unique description of their dynamical behavior in terms of a *nondeterministic* and *maximally parallel* application of rules. The first of these concepts puts the focus not on an actual realization of behavior of a P system, but on all possible computations possible with it, i.e. on the (formal) *language* generated by it. The concept of maximal parallelism allows interesting control structures, but seems rather inappropriate when modelling in a biological context.

Therefore, a number of researchers have turned to *dynamical* P system models, where the nondeterministic dynamics is replaced by a sequential and probabilistic evolution law. Two important approaches are *dynamically probabilistic P systems* [32] and the *metabolic algorithm* developed and propagated by V. Manca and colleagues [4,25]. The first is directly based on mass action kinetics [15], whereas the latter considers a special form of competition of rules for objects, called the *mass partition principle*. Another approach has been proposed in [33], where rules have fixed reaction rates.

As has been discussed in [27], static[9] dynamical P systems are Markov chains. Unfortunately, the *state space* of a dynamical P system is usually very large.

Example 1. (Brusselator) Consider a membrane system on a set of three objects $O = \{A, B, C\}$, and with the following four rules:

$$r_1 : \quad \lambda \to A, \qquad\qquad r_2 : \quad A \to B,$$
$$r_3 : \quad 2A + B \to C, \qquad\qquad r_4 : \quad C \to 3A,$$

where λ denotes the empty multiset and can be interpreted as a (yet unspecified) inflow. This is an equivalent of the Brusselator, a discrete model of the Belousov-Zhabotinskii chemical oscillation reaction. Its state space X consists, in principle, of all multiset configurations on three objects, i.e. $X = \mathbb{N}_0^3$, where we identify a multiset with a vector of nonnegative numbers $\mathbb{N}_0 = \{0, 1, 2, \dots\}$.

Although no dynamical law has yet been specified, a biochemically motivated law, based on mass action kinetics, for example, results in a transition matrix P_{ij}, where $i, j \in X$ and the entries of P_{ij} depend, usually nonlinearly, on the state i.

The *asymptotic behavior* of a dynamical P system can then, in principle, be defined as its stationary distribution as in the case of the webgraph. Applying ideas from the *PageRank* algorithm, halting states of the system, in analogy with dangling links, are assumed to result in a teleportation to an arbitrary state of X. Also, at each time step there is a small probability $\alpha > 0$ that the system teleports to a random state from X.

It should be clear that a prerequisite for the definition of the asymptotic behavior of a dynamical P system is the *finiteness* of its state space, so this concept will only be applicable in some special cases. Even then, due to the potential size of the state space it is not clear at the moment how to actually perform such an analysis. In the next section we will therefore propose a much simpler characterization of the "asymptotic behavior" of a dynamical P system.

[9] A P system is static if the membrane structure does not evolve in the course of time. To be more precise: membrane creation or destruction are not allowed in a static P system.

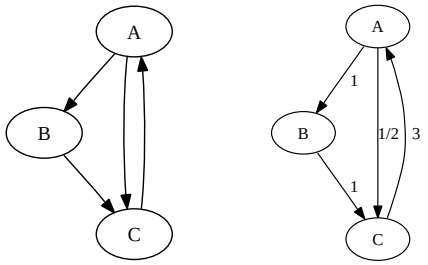


Fig. 5.1 The object network (left panel) and the stoichiometric network (right panel) of the Brusselator example.

5 Approximating asymptotic behavior

Since the state space in dynamical P systems is usually infinite, a stationary distribution usually does not exist, or even if it does, it is not clear how to actually compute it. An interesting alternative is to simplify the situation considerably. Instead of working with the state space on which the dynamics takes place, we work with the *object network* of the P system.

Definition 1 *The* object network *of a P system is the directed graph $D = (V, A)$, where the vertex set V is given by the set of objects, and there exists an arc $(u, v) \in A$ between two objects $u, v \in V$ if there exists a rewriting rule of the form*

$$p_1 u_1 + \cdots + p_n u_n \to q_1 v_1 + \cdots + q_m v_m, \qquad p_k, q_l \geq 1 \text{ for all } k \leq n, l \leq m,$$

and furthermore $u = u_i$ and $v = v_j$ for some indices $i \leq n$ and $j \leq m$.

The *connectivity matrix* of a P system is then the adjacency matrix of its object network.

Definition 2 *The* ranking matrix *of a P system is the matrix $\bar{\bar{C}}$ (confer Eq. 49) where C is its connectivity matrix.*

Example 2 The object network of the Brusselator example is shown in the left panel of Figure 5.1. It is obviously aperiodic and irreducible. Its connectivity matrix and the corresponding ranking matrix are, respectively,

$$C = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \qquad \bar{\bar{C}} = \begin{pmatrix} \alpha/3 & 1/2 - \alpha/6 & 1/2 - \alpha/6 \\ \alpha/3 & \alpha/3 & 1 - 2\alpha/3 \\ 1 - 2\alpha/3 & \alpha/3 & \alpha/3 \end{pmatrix}.$$

Definition 3 *The* stationary object distribution *of a P system is the dominant eigenvector of its ranking matrix.*

Numerical values are given with an accuracy of three digits in the rest of the paper, if not otherwise stated.

Example 3 For the Brusselator example, choosing $\alpha = 0.01$, the stationary object distribution is

$$(0.399, 0.201, 0.399)^t,$$

i.e. in the long-term limit one expects 40 percent of all objects to be of type A, 20 percent to be of type B, and 40 percent to be of type C. Of course, depending on the actual dynamical law, the true long-term distributions of these numbers will vary considerably from these approximate values.

Up to now, only the *topological* information about how objects can be transformed into each other has been used. However, in a P system there are two more levels that can be considered, namely (i) the stoichiometry and (ii) reaction rates.

Definition 4 *The* stoichiometric network *of a P system is the weighted digraph $W = (V, A, w)$, where (V, A) is the object network and w is the* stoichiometric weight *on each arc, i.e. $w(u, v) \in \mathbb{Z}$ describes the number of objects of type $v \in V$ that result when all the rules are applied simultaneously where $u \in V$ appears (somewhere) on the left-hand side, and v (somewhere) on the right-hand side.*

Note that, to avoid multigraphs with parallel arcs, the definition of the stoichiometric network effectively sums the contributions of all rules to the stoichiometry of a target object. Normalizing the stoichiometric weights, we arrive at the *normalized stoichiometric weights* $W^{(1)}$, conveniently written as a matrix whose rows all sum to one:

$$W^{(1)}_{uv} := \frac{W_{uv}}{\sum_{w \in V} W_{uw}}.$$

Definition 5 *The* stoichiometric ranking matrix *of a P system is the matrix $\bar{\bar{W}}^{(1)}$ (confer Eq. 49).*

Example 4 The stoichiometric weights, the normalized stoichiometric weights, and the stoichiometric ranking matrix of the Brusselator example are, respectively

$$W = \begin{pmatrix} 0 & 1 & 1/2 \\ 0 & 0 & 1 \\ 3 & 0 & 0 \end{pmatrix}, \quad W^{(1)} = \begin{pmatrix} 0 & 2/3 & 1/3 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix},$$

$$\bar{\bar{W}}^{(1)} = \begin{pmatrix} \alpha/3 & 2/3 - \alpha/6 & 1/3 - \alpha/6 \\ \alpha/3 & \alpha/3 & 1 - 2\alpha/3 \\ 1 - 2\alpha/3 & \alpha/3 & \alpha/3 \end{pmatrix}.$$

The latter has dominant eigenvector $p \approx (0.375, 0.25, 0.375)^t$.

One important question, that we have avoided so far, is the behavior of stationary distributions of a (weighted) digraph under restriction. In applications, we might only be interested in a small part of a P system, or the dynamics of the complement might not even be known explicitly. To address this question, we interprete the stationary distribution over the objects as an *invariant flow*. Given a ranking matrix R and a stationary distribution p for R, its nonnegative entries R_{uv} multiplied by the (probability) mass p_u are interpreted as a flow from node u to node v. The total sum of inflows $\sum_{w \in V}(p_w R_{wu})$ into a node $u \in V$, and the total sum of outflows $p_u \cdot \sum_{w \in V} R_{uw}$ are equal. Moreover, the total flow $\sum_{u,v \in V}(p_u R_{uv})$ is equal to one.

Given a *cut* of V, i.e. a set $A_1 \subset A$ of arcs that separates $V_1 \subset V$ from its complement $V \setminus V_1$, the corresponding partition of arcs is $[A_1, A \setminus A_1]$, where $A_1 = \{(u, v) \mid$ exactly one of u or v lies in $V_1\}$. We interprete V_1 as a subnetwork of V. It is clear that the invariant in- and outflows across the cut are equal:

$$\sum_{(u,v) \in A_1, u \notin V_1} p_u R_{uv} = \sum_{(u,v) \in A_1, u \in V_1} p_u R_{uv}.$$

If we replace $V \setminus V_1$ by a single node $e \notin V$ and all relevant arcs by their obvious connections with e, the invariant distribution of the network $V_1 \cup \{e\}$, restricted to V_1 and then normalized, will be the same as that of V for a certain *effective* choice of in-weights $w(e, V_1)$.

However, to compute these effective in-weights, one needs to know the structure of the complement of V_1 — or one has to treat the weights $w(e, V_1)$ as unknowns and use further constraints to determine them.

Example 5 Consider the network shown in the left panel of Figure 5.2. Its dominant eigenvector is $p = (0.300, 0.150, 0.275, 0.275, 0.075, 0.125)^t$.

The dotted arcs represent a cut separating the subnetwork $V_1 = \{A, B, C\}$ from $\{D, E, F\}$. Replacing the latter by the single node S results in the network shown in the right panel of Figure 5.2. With the effective edge weights from the figure, $w(S, A) = 1/6$ and $w(S, C) = 5/6$, its dominant eigenvector is $p' = (12/35, 6/35, 11/35, 6/35)^t$. The

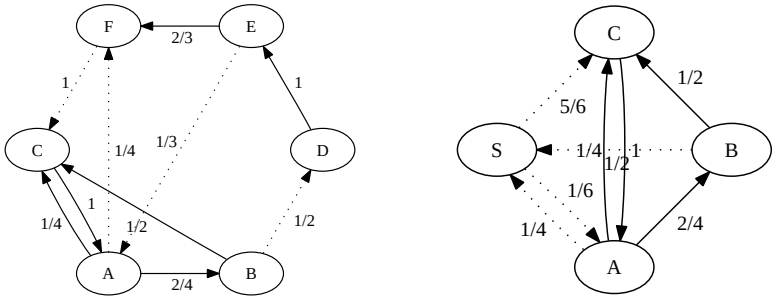


Fig. 5.2 Extended Brusselator system, where three additional objects have been added. The numbers on the edges are the normalized stoichiometric weights. The network in the right panel is derived from the left one by the construction described in the text.

(normalized) restriction of p to V_1 is $p_1 = (0.414, 0.207, 0.379)^t$, which is the same as the restriction of p' to V_1.

More important is the situation in which we are given the flows across the cut as an external constraint. Normalizing the sum of inflows such that $\sum_{(u,v)\in A_1, u\notin V_1} p_u R_{uv} = 1$, we can interprete this flow as a probability flow. It is then possible to calculate an invariant distribution for the subnetwork by closing the system, connecting in- with outflows. Of course, to avoid the reducibility and aperiodicity problems, we need to introduce some teleportation, as before.

Let us also briefly consider the analysis of steady state fluxes in biochemical networks. Given a stoichiometric matrix $S \in \mathbb{Z}^{m\times n}$ that describes the possible transitions of a chemical system, and some external fluxes $b \in \mathbb{R}^m_+$, one searches for a solution $x \in \mathbb{R}^n$ of the equation $S \cdot x = 0$, which is interpreted as a steady state flux. In the context of P systems, we can think of x as an *application vector*, telling us how often each rule has to be used. Unfortunately, linear algebra cannot be used, since the solutions need to be positive, i.e. it is necessary that $x_i \geq 0$ for some of the components of $x = (x_1, \ldots, x_n)$, since we cannot have negative rule applications. Therefore, one resorts to convex analysis and calculates the convex cone of all possible solutions [20]. This cone is usually not unique, so there are many possible steady state fluxes across the system.

But consider now what happens if we make use of the probabilities for transitions, not only the topology. The invariant distribution then induces a *unique* steady state flux, in contrast to the topological case. The implications of this, especially with regard to pathway analysis, have yet to be fully realized (see below).

6 A new complexity measure

From a stationary distribution of the object network we immediately derive a complexity measure for P systems.

Definition 6 *The* object entropy *of a P system is the entropy of its stationary object distribution. That is, if $p \in \mathbb{R}^n_+$ is its stationary object distribution, then*

$$h = \frac{-\sum_{i=1}^{n} p_i \log p_i}{\log n}$$

is its object entropy, where $0 \log 0$ is interpreted as 0.

The denominator has been chosen such that $0 \leq h \leq 1$ holds. A low value of h signifies a very ordered state, whereas a value of h close to one signifies that the stationary object distribution is almost uniform.

Example 6 For the Brusselator example, the object entropy is $h = 0.961$. For the extended Brusselator example in the left panel of Figure 5.2, the object entropy is $h = 0.921$, corresponding to the higher complexity of it. The system on the right panel of that figure has object entropy $h = 0.963$.

This idea generalizes the *global entropy* of [9], where a similar complexity measure has been introduced for probabilistic P systems with an evolution tree. Of course, the question arises what the advantage of such a measure is, compared to other complexity measures (for a list of possible candidates, see [7]). An important point here is that the definition of entropy of an invariant distribution is a mathematically elegant concept that quantifies the complexity of the dynamics of a P system in a way that easily relates to complexity considerations in other fields of science (confer [2]).

7 Page ranking in P system identification

In a previous work [26] we have discussed the general problem of identification of P systems; here we will focus on the application of page ranking to this problem. System identification can be considered the reverse of the usual modelling and analysis process. Instead of analyzing a *given* P system, the problem is to *find* an interesting P system that then can be analysed, for example by simulation studies. This is particularly interesting in the application of P systems to biochemical systems. To this extent, public databases on the internet can be used that store and collect information about biochemical reactions. These include WIT, EcoCyc, MetaCyc [19], aMAZE and KEGG [18]. We will only consider the LIGAND database here [14], which is a particular database inside

the KEGG repository. As of version 42.0, LIGAND contains information about 15053 chemical compounds (KEGG COMPOUND), 7522 biochemical reacions (KEGG REACTION) and 4975 enzymes (KEGG ENZYME) in ASCII text files that are easily parseable by computer.

In the usual approach [10, 28] one constructs an *undirected* metabolite network graph $G = (V, E)$ from these files, where nodes represent compounds, and edges represent reactions (for simplicity, we do not consider enzymes here). Two compounds $u, v \in V$ in the metabolite graph are connected by an arc $(u, v) \in E \subseteq V^2$ if there exists a reaction in which both u and v participate. Note that u and v can both occur on the same side of a reaction, in contrast to what we have done for P system object networks, resulting in an undirected as opposed to a directed graph. The main problem considered in the bioinformatics community is the extraction of (meaningful) possible pathways that allow to transform one compound $s \in V$ into a target compound $t \in V$, which is equivalent to the k *shortest path problem* [11].

A particular problem with this approach is the existence of so-called *currency* metabolites [16]. These are usually small biomolecules that participate in a large number of reactions, and are used to store and transfer energy and/or certain ions. Examples of currency metabolites include $\mathrm{H_2O}$, ATP, and NADH. Because of them, for example, there exist more than 500000 distinct pathways of length at most nine between glucose and pyruvate [22], most of which are not biochemically feasible. The solution considered by Croes and co-workers is to weight the paths by the (out-) degrees of their vertices, such that vertices with a large degree are punished relative to compounds with a higher specifity, i.e. a lower degree [10].

Here we propose to use a *directed* metabolite graph that more realistically captures the flow constraints of the biochemical reaction network, and to use the stationary distribution of such a biochemical object network to weight the paths. Currency metabolites are expected to have a large stationary probability, since they partake in many circular reaction patterns, and interesting pathways should then be found more effectively by bounding the total path weight.

We shall demonstrate this here by way of a simple example.

Example 7 Consider the network graph in the left panel of Figure 7.3. This has been generated by starting at Caffeine (C007481) in the KEGG COMPOUND database. All paths of length at most 2 have been generated, and the resulting graph has been pruned, such that no leaves remains (i.e. such that no vertices remain with only one arc). The resulting network is shown in Figure 7.3 on the right. From this, the stoichiometric network graph on the left has been created. For each pair of compounds $(u, v) \in V$, we sum over the contributions to the stoichiometry from all rules. These weights are then normalized, such that they can be interpreted as transition probabilities, and are displayed in the left figure. The stationary eigenvector (using $\alpha = 0.01$) of the stoichiometric

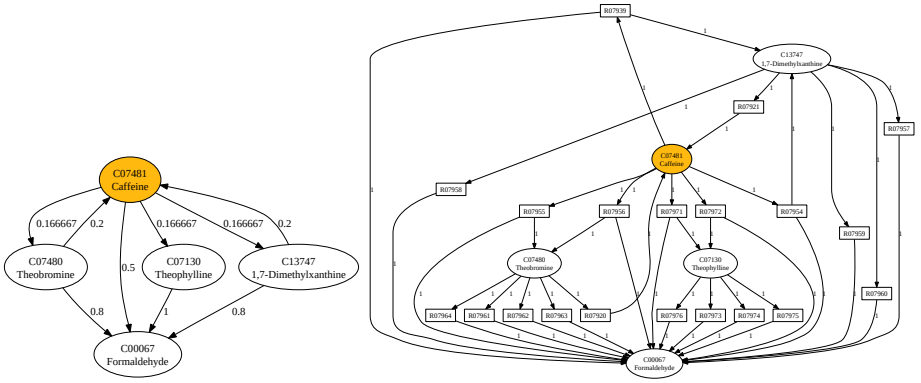


Fig. 7.3 Normalized stoichiometric network graph (left panel) for Caffeine example in the text. The right panel shows the underlying reaction network in a (place-transition) Petri net representation, where boxes correspond to distinct chemical reactions.

ranking matrix is

$$p = (0.487, 0.122, 0.122, 0.146, 0.122)^t,$$

corresponding to the compounds

$$(\mathrm{C00067}, \mathrm{C07130}, \mathrm{C07480}, \mathrm{C07481}, \mathrm{C13747}),$$

and we see that Formaldehyde (C00067) qualifies as a currency compound. In KEGG REACTION there are 57 reactions in which it participates, in fact. Note that the stoichiometric network graph where paths of length up to three are allowed cannot be drawn sensibly anymore, since it already contains 58 vertices, many of which are due to reactions with Formaldehyde. The invariant distribution can be calculated easily, however.

P systems identification is then possible by first generating a large stoichiometric network graph, calculating its invariant distribution $p \in \mathbb{R}_+^N$, and using its components p_i, $1 \leq i \leq N$, to define weights $N \cdot p_i$ for a second pathway search (the constant N is used to ensure that the average weight is one). Only compounds encountered on paths with a weight below a certain, user-defined threshold are then used to define a P system model that captures the (hopefully) relevant biochemical reactions.

8 Discussion

In this paper we have shown some applications of page ranking to the analysis and identification of P systems. Dynamical P systems can be considered Markov chains, and Google's page ranking then corresponds to the stationary eigenvector of the transition

matrix, after adding a small positive constant to ensure irreducibility and aperiodicity. For P systems, page ranking allows to define a probability distribution on the objects (and, dually, also on the rules), and this in turn allows to define the entropy of a P system, generalizing ideas of [9]. A different application has been in the identification of P system models from biochemical databases. There, the invariant distribution should allow to search more effectively for pathways, improving the degree weights introduced by Croes and co-workers. Although the complete stoichiometric graph available in the LIGAND database is quite large (more than 10000 vertices), the eigenvector calculation has to be done only once. The test of this idea is underway.

We have also presented some interesting problems and directions for future work. Lastly, let us remark that it is possible to generalize the page ranking approach to systems with an infinite state space (by normalizing the *average* page rank, and not the sum of all ranks). This should be especially interesting for the analysis of (pseudo-) lattice digraphs, being a continuation of the work in [27]. More generally, this work was motivated by the urge to adapt the methods of dynamical systems theory to P systems, and from this perspective the invariant distribution of a P system can be considered to represent the asymptotical dynamical behavior of a given system. In particular, we can now give operational definitions of the concept of "stable fixed points" for P systems as states with a large invariant probability, whereas "transient" states will have a very small invariant probability (on the order of α).

Acknowledgements. This work has been supported by the Dutch Science Foundation (NWO) under grant no. 635.100.006.

Bibliography

[1] Altman, A., Tennenholtz., M.: Ranking systems: the PageRank axioms. Electronic Commerce, Proceedings of the 6th ACM conference on Electronic commerce (2005) 1–8
[2] Badii, R., Politi, A.: *Complexity. Hierarchical structures and scaling in physics.* Cambridge University Press (1997)
[3] Bapat, R. B., Raghavan, T. E. S.: *Nonnegative Matrices and Applications.* Cambridge University Press (1997)
[4] Bianco, L., Fontana, F., Manca, V.: P systems with reaction maps. Int. J. Found. Comp. Sci. **17** (2006) 3–26
[5] Brémaud, P.: *Markov Chains. Gibbs Fields, Monte Carlo Simulation, and Queues*. Springer-Verlag (1999)
[6] Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems **30** (1998) 107–117
[7] Chakrabarti, D., Faloutsos, C.: Graph Mining: Laws, Generators, and Algorithms. ACM Computing Surveys **38** (2006) 1–69

[8] Ciobanu, G., Păun, G., Pérez-Jiménez, M.-J. (Eds.) *Applications of Membrane Computing*. Springer-Verlag (2006)
[9] Cordón-Franco, A., Sancho-Caparrini, F.: A note on complexity measures for probabilistic P systems. Journal of Universal Computer Science **10** (2004) 559–568
[10] Croes, D., Couche, F., Wodak, S. J., van Helden, J.: Inferring Meaningful Pathways in Weighted Metabolic Networks. J. Mol. Bio. **356** (2006) 222–236
[11] Epstein, D.: Finding the k Shortest Paths. SIAM J. Comput. **28** (1998) 652–673
[12] Golub, G. H., Van Loan, C. F.: *Matrix Computations.* Johns Hopkins University Press (1996)
[13] Google: Google Technology, `http://www.google.com/technology/`
[14] Goto, S., Nishioka, T., Kanehisa, M.: LIGAND: chemical database for enzyme reactions. Bioinformatics **14** (1998) 591–599
[15] Heinrich, R., Schuster, S.: *The Regulation of Cellular Systems.* Springer-Verlag (1996)
[16] Huss, M., Holme, P.: Currency and commodity metabolites: their identification and relation to the modularity of metabolic networks. IET Syst. Biol. **1** (2007) 280–285
[17] Kamvar, S., Haveliwala, T., Golub, G.: Adaptive methods for the computation of PageRank. Linear Algebra Appl. **386** (2004) 51–65
[18] Kanehisa, M., Araki, M., Goto, S., Hattori, M., et al.: KEGG for linking genomes to life and the environment. Nucleic Acids Research **36** (2008) D380–D484. URL: `http://www.genome.jp/kegg/`
[19] Karp, P. D., Riley, M., Saier, M., Paulsen, I. T., et al.: The EcoCyc and MetaCyc databases. Nucleic Acids Research **28** (2000) 56–59. URL: `http://metacyc.org/` and `http://biocyc.org/`
[20] Kauffman, K. J., Prakash, P., Edwards, J. S.: Advances in flux balance analysis. Current Opinion in Biotechnology **14** (2003) 491–496
[21] Klipp, E., Herwig, R., Kowald, A., Wierling, C., et al.: *Systems Biology in Practice*. Wiley-VCH (2005)
[22] Kuffner, R., Zimmer, R., Lengauer, T.: Pathway analysis in metabolic databases via differential metabolic display (DMD). Bioinformatics **16** (2000) 825–836
[23] Langville, A. N., Meyer, C. D.: Deeper inside PageRank. Internet Math. **1** (2004) 335–380
[24] Ma, N., Guan, J., Zhao, Y.: Bringing PageRank to the citation analysis. Information Processing & Management **44** (2008) 800–810
[25] Manca, V., L. Bianco, L.: Biological networks in metabolic P systems. BioSystems **91** (2008) 489–498
[26] Muskulus, M. Identification of P system models assisted by biochemical databases. In: Ibarra, O. H., Sosík, P. (Eds.) Prague International Workshop on Membrane Computing, Preliminary proceedings, Silesian University in Opava, Faculty of Philosophy and Science (2008) 46–49
[27] Muskulus, M., Besozzi, D., Brijder, R., Cazzaniga, P., et al.: Cycles and communicating classes in membrane systems and molecular dynamics. Theor. Comp. Sci.

372 (2007) 242–266
[28] Noirel, J., Ow, S. Y., Sanguinetti, G., Jaramillo, A., et al.: Automated extraction of meaningful pathways from quantitative proteomics data. Briefings in Functional Genomics and Proteomics (2008), in press
[29] Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the web. Technical Report, Stanford University (1998). Available from: `http://dbpubs.stanford.edu:8090/pub/1999-66`
[30] Păun, G.: Computing with Membranes. J. Comput. Syst. Sci. **61** (2000) 108–143
[31] Păun, G.: *Membrane computing. An Introduction.* Springer–Verlag (2002)
[32] Pescini, D., Besozzi, D., Mauri, G., Zandron, C.: Dynamical Probabilistic P Systems. Int. J. Found. Comp. Sci. **17** (2006) 183–204
[33] Romero-Campero, F. J., Pérez-Jiménez, M. J.: Modelling gene expression control using P systems: The Lac Operon, a case study. Biosystems **91** (2008) 438–457
[34] Vise, D., Malseed, M.: *The Google Story.* Random House (2006).

An algorithm for nondeterministic object distribution in P systems and its implementation in hardware

Van Nguyen, David Kearney, Gianpaolo Gioiosa

University of South Australia, School of Computer and Information Science,
Mawson Lakes Boulevard, Adelaide, 5095, Australia
{Van.Nguyen, David.Kearney, Gianpaolo.Gioiosa}@unisa.edu.au

We have recently developed a prototype hardware implementation of membrane computing using reconfigurable computing technology. This prototype, called Reconfig-P, exhibits a good balance of performance, flexibility and scalability. However, it does not yet implement nondeterministic object distribution. One of our goals is to incorporate nondeterministic object distribution into Reconfig-P without compromising too significantly its performance, flexibility or scalability. In this paper, we (a) propose an algorithm for nondeterministic object distribution in P systems, and (b) describe and evaluate a prototype hardware implementation of this algorithm based on reconfigurable computing technology. The results of our evaluation of the prototype implementation show that our proposed algorithm can be efficiently implemented using reconfigurable computing technology. Therefore there is strong evidence that it is feasible to incorporate nondeterministic object distribution into Reconfig-P as desired.

1 Introduction

We have recently developed a prototype hardware implementation of membrane computing using reconfigurable computing technology. This prototype, called Reconfig-P, exhibits a good balance of performance, flexibility and scalability. However, it does not yet implement nondeterministic object distribution. One of our goals is to incorporate nondeterministic object distribution into Reconfig-P without compromising too significantly its performance, flexibility or scalability. In this paper, we (a) propose an algorithm for nondeterministic object distribution in P systems, and (b) describe and evaluate a prototype hardware implementation of this algorithm based on reconfigurable computing technology. The results of our evaluation of the prototype implementation show that our proposed algorithm can be efficiently implemented using reconfigurable computing technology. Therefore there is strong evidence that it is feasible to incorporate nondeterministic object distribution into Reconfig-P as desired.

The contents of the paper are as follows. In Section 2, we discuss the background to the research problem. In Section 3, we state the research problem. In Section 4, we present and explain our proposed algorithm for nondeterministic object distribution in P systems. In Section 5, we evaluate the correctness and theoretical efficiency of the algorithm. In Section 6, we describe our prototype hardware implementation of the algorithm. In Section 7, we present and discuss empirical results related to the efficiency of the prototype implementation. Finally, in Section 8, we draw some conclusions regarding the significance of our research results.

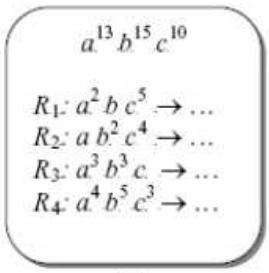
2 Background

In this section, we present the background to the problem that motivates the research described in this paper. First, we present a brief overview of membrane computing. Second, we present formal definitions of nondeterminism and maximal parallelism in the context of object distribution in P systems. Third, we discuss our overall research program, which thus far has produced a prototype hardware-based computing platform for membrane computing called Reconfig-P. Finally, we introduce the reconfigurable computing technology used in Reconfig-P.

2.1 Membrane computing Membrane computing is a branch of bio-inspired computing. It investigates models of computation inspired by certain structural and functional features of biological cells, especially features that arise because of the presence and activity of biological membranes. There are several types of membrane computing models. These include cell-like models, tissue-like models and neural models. As many of the fundamental features of membrane computing models are present in cell-like models, we focus on cell-like models in our research.

Biological membranes define compartments inside a cell or separate a cell from its environment. The compartments of a cell contain chemical substances. The substances within a compartment may react with each other or be selectively transported through the membrane surrounding the compartment (e.g., through protein channels) to another compartment as part of the cell's complex operations.

In a membrane computing model, called a *P system*, multisets of objects (chemical substances) are placed in the regions defined by a hierarchical membrane structure, and the objects evolve by means of reaction rules (chemical reactions) also associated with the regions. The reaction rules are applied in a maximally parallel, nondeterministic manner (in a sense to be defined below). The objects can interact with other objects inside the same region or pass through the membrane surrounding the region to neighbouring regions or the cell's environment. These characteristics are used to define transitions between configurations of the system, and sequences of transitions are used to define computations. A computation halts when for every region it is not possible to apply any reaction rule.



(a)

R_1	R_2	R_3	R_4
0	0	0	3
0	0	1	2
0	0	3	1
0	1	0	2
0	1	2	1
0	1	4	0
0	2	2	0
1	0	2	1
1	0	3	0
1	1	1	0
2	0	0	0

(b)

Fig. 2.1 (a) A P system with one region and four reaction rules, and (b) a list of the possible ways in which the reaction rules in the P system can be applied in the current transition given that they must be applied in a maximally parallel manner.

For more information about the fundamentals of membrane computing, we refer the reader to [6].

2.2 Nondeterministic, maximally parallel application of reaction rules in P systems Nondeterminism and maximal parallelism are key features of P systems. We define these features below, first informally and then formally.

Definition of maximal parallelism In a transition of a P system, if any reaction rule can be applied, it *must* be applied. That is, in a transition, if a reaction rule is able to consume a multiset of objects that have not been consumed by other reaction rules, then it must consume that multiset of objects. This is the *maximal parallelism* property of P systems.

Figure 2.1 shows an example P system, and lists the possible ways in which the reaction rules in the P system's only region can be applied given that they must be applied in a maximally parallel manner. The actual way in which the reaction rules are applied is selected nondeterministically (in a sense to be defined below) from this range of possibilities.

Suppose that zero instances of R_1, one instance of R_2, one instance of R_3 and one instance of R_4 were to be applied in the current transition. This would *not* be a maximally parallel application of the reaction rules. This is because, after the application of the reaction rules, the multiset of objects remaining in the region would have been $a^5b^5c^2$, in which case an additional instance of R_3 could have been applied.

Note that the case in which the number of instances for each reaction rule in a region is zero is regarded as an application of the reaction rules.

Definition of nondeterminism As illustrated in Figure 2.1, when the reaction rules in a region are applied in a maximally parallel manner during a transition, there are often multiple ways in which the objects in the region can be distributed to the reaction rules. (For instance, in the current transition of the P system depicted in Figure 2.1, the objects can be distributed in eleven different ways, so there are eleven possible maximally parallel applications of the reaction rules.) If the maximally parallel application that actually occurs is selected from this range of possibilities at random, we say that the application is *nondeterministic*. Otherwise, we say that the application is *deterministic*.

Formal definitions We now present formal definitions of maximal parallelism and nondeterminism.

Let the $n \geq 1$ reaction rules in a region of a P system be:

R_1: $o_1^{a_{11}} o_2^{a_{21}}$ ... $o_m^{a_{m1}} \rightarrow$ ...

R_2: $o_1^{a_{12}} o_2^{a_{22}}$ ... $o_m^{a_{m2}} \rightarrow$ ...

...

R_n: $o_1^{a_{1n}} o_2^{a_{2n}}$ ... $o_m^{a_{mn}} \rightarrow$ ...,

where o_1, o_2, ..., o_m are the object types in the P system, and each a_{ij} $(1 \leq i \leq m, 1 \leq j \leq n)$ is a nonnegative integer. Let the multiset of objects available in the region be $o_1^{b_1} o_2^{b_2} ... o_m^{b_m}$, where each b_i $(1 \leq i \leq m)$ is a nonnegative integer. And let x_1, x_2, ..., x_n denote the numbers of instances of the reaction rules R_1, R_2, ..., R_n, respectively, to be applied in the current transition.

Suppose that the n reaction rules in the region are applied during a transition. As it is not possible to consume more objects of a given type than are initially available, each of the following conditions must be satisfied:

$a_{11}x_1 + a_{12}x_2$ + ... + $a_{1n}x_n \leq b_1$

$a_{21}x_1 + a_{22}x_2$ + ... + $a_{2n}x_n \leq b_2$

...

$a_{m1}x_1 + a_{m2}x_2$ + ... + $a_{mn}x_n \leq b_m$

Definition of maximal parallelism Let s = (s_1, s_2, ..., s_n) be a solution to the linear system of inequalities given above, where s_1, s_2, ..., s_n are nonnegative integers. This solution represents values of x_1, x_2, ..., x_n for which all of the inequalities in the lin-

ear system hold. Then s corresponds to a maximally parallel application of the reaction rules R_1, R_2, ..., R_n if and only if the solution $v = (v_1, v_2, ..., v_n)$ (where $v_1, v_2, ..., v_n$ are nonnegative integers) of the linear system

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leq b'_1$$

$$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \leq b'_2$$

$$...$$

$$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leq b'_m,$$

where

$$b'_1 = b_1 - a_{11}s_1 - a_{12}s_2 - ... - a_{1n}s_n$$

$$b'_2 = b_2 - a_{21}s_1 - a_{22}s_2 - ... - a_{2n}s_n$$

$$b'_m = b_m - a_{m1}s_1 - a_{m2}s_2 - ... - a_{mn}s_n$$

is such that v is the zero vector.

(Note that if $v_1, v_2, ..., v_n$ were real numbers rather than integers, we would have that each of these values is greater than or equal to zero and less than 1.)

Definition of nondeterminism There are $p \geq 0$ possible values for s. If the value for s is selected at random from this range of p possibilities, then s corresponds to a nondeterministic maximally parallel application of the reaction rules. Otherwise, it corresponds to a deterministic maximally parallel application of the reaction rules.

2.3 Our overall research program To exploit the performance advantage of the large-scale parallelism of P systems, it is necessary to execute them on a parallel computing platform. To this end, researchers have investigated parallel computing platforms for membrane computing, including platforms based on software (see, e.g., [1]) and platforms based on hardware (see, e.g., [4], [5] and [7]).

Our overall research program is focused on hardware-based parallel computing platforms for membrane computing. Hardware-based parallel computing platforms execute algorithms that have been directly implemented in hardware. The hardware platform implements the algorithm in terms of the parallel activities of a certain number of processors that are spatially, rather than temporally, related. The ability to use parallel processors brings a potentially very significant improvement in execution time performance. However, the use of the spatial dimension means that the number of processors, and

therefore the class of algorithms, that can be implemented on the platform is constrained by the amount of hardware resources available on the platform.

Our research involves an investigation of a certain novel approach to the development of a parallel computing platform for membrane computing. This approach involves the use of reconfigurable hardware and an intelligent software component that is able to configure the hardware to suit the specific properties of the P system to be executed. We have developed a prototype computing platform called Reconfig-P based on the approach [4] [5]. Reconfig-P is currently able to execute P systems that are the same as basic cell-like P systems, except that objects are distributed to reaction rules in a deterministic (rather than a nondeterministic) manner. It is the first hardware-based parallel computing platform for membrane computing to implement parallelism at both the system and region levels, and is one of the most complete hardware implementations of membrane computing published to date.

The implementation approach on which Reconfig-P is based involves

- use of a reconfigurable hardware platform,
- generation of a customised digital circuit for each P system to be executed, and
- use of a hardware description language that allows digital circuits to be specified at a level of abstraction similar to the level of abstraction at which a general-purpose procedural software programming language (such as C) allows algorithms to be specified.

In the approach, a software component of the computing platform is responsible for analysing the structural and behavioural features of the P system to be executed and producing a hardware description for the P system that is tailored to these features. When determining the hardware description for the P system, the software component aims to maximise performance and minimise hardware resource consumption. The empirical results presented in [4] show that for a variety of P systems Reconfig-P achieves very good performance while making economical use of hardware resources.

The natural next step in the development of Reconfig-P is to attempt to incorporate nondeterministic object distribution into Reconfig-P. Although it is quite clear that supporting nondeterminism will result in higher hardware resource consumption and a degradation in execution time performance, it is unknown just how efficient an implementation of nondeterministic object distribution could be made to be. Therefore, despite the positive results mentioned above, it is unknown whether nondeterministic object distribution could be incorporated into Reconfig-P without compromising too significantly its performance, flexibility and scalability.

2.4 Reconfigurable computing technology As already mentioned, hardware-based computing platforms execute algorithms that have been directly implemented in hardware. In one approach, an application-specific integrated circuit (ASIC) is used. The

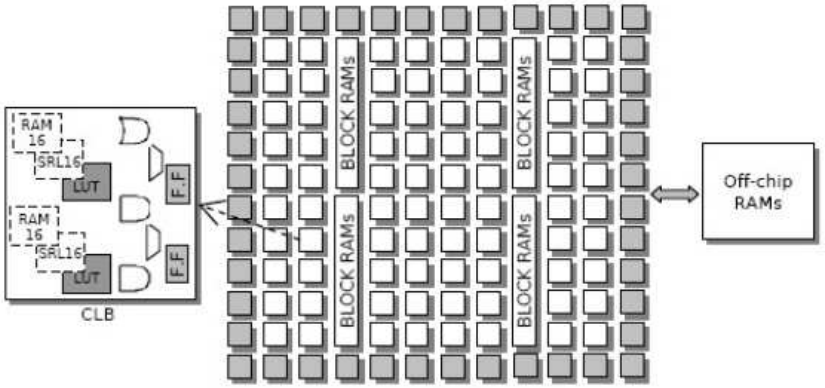


Fig. 2.2 An FPGA chip with different types of memory.

design of an ASIC is tailored to a specific algorithm. As a consequence, ASICs usually achieve a higher performance than software-programmed microprocessors when executing the algorithm for which they were designed. However, with this higher performance comes reduced flexibility: as the implemented algorithm is fabricated on a silicon chip, it cannot be altered without creating another chip. In another approach, reconfigurable hardware is used. Unlike ASICs, reconfigurable hardware can be modified. Therefore, by using reconfigurable hardware, it is possible to improve on the performance of software-based computing platforms while retaining some of their flexibility. The computing paradigm based on the use of reconfigurable hardware is known as *reconfigurable computing*. We now briefly introduce the reconfigurable computing technology used in Reconfig-P.

FPGAs A field-programmable gate array (FPGA) is a type of reconfigurable hardware device. A standard FPGA consists of a matrix of configurable logic blocks (CLBs). The CLBs are connected by means of a network of wires. They can be used to implement logic or memory. Each CLB is composed of a number of slices, each of which consists of two four-input lookup tables (LUTs), two flip-flops and some internal logic (e.g., carry logic used in the implementation of arithmetical operations). The LUTs can be used to implement logic gates or small memories. The flip-flops can be used to create state machines. The CLBs at the periphery of the FPGA can perform I/O operations. The functionality of the CLBs and their interconnections can be modified by loading configuration data from a host computer. In this way, any custom digital circuit can be mapped onto the FPGA, thereby enabling the FPGA to execute a variety of applications.

Figure 2.2 shows an FPGA chip with different types of memory.

Handel-C Digital circuits are specified in hardware description languages. A popular high-level hardware description language is Handel-C. Handel-C allows a hardware cir-

cuit to be specified at the algorithmic level rather than at a level at which the structure of the circuit is apparent, and therefore eases the process of designing a circuit for an application. Handel-C includes a small subset of the C programming language as well as additional constructs for the configuration of a hardware device, including constructs related to parallelism, communication, timing and bit manipulation. Every Handel-C statement takes exactly one clock cycle to execute. Therefore it is relatively easy for programmers to measure the number of clock cycles that it takes to execute a particular algorithm on the hardware device. In terms of data storage, Handel-C provides several options, including arrays, on-chip distributed RAMs, on-chip dedicated RAMs and off-chip RAMs. Each of the storage options has its own benefits and drawbacks, depending on the specific data to be stored. We discuss the storage options below.

Data storage

Arrays An array in Handel-C is a collection of individual registers, each of which is implemented directly as one or more flip-flops. An element of an array may be used exactly like an individual register (by means of an index into the array) and therefore all elements of the array can be accessed in parallel. However, this is achieved by means of a multiplexer between the registers, which can be expensive to implement, both in terms of the hardware resources required and in terms of the logic delay induced.

On-chip distributed RAMs The LUTs in CLBs can be configured to operate as 16 $\times$ 1 RAMs (i.e., 16-deep, 1-bit-wide RAMs). In this way, RAMs can be placed at arbitrary locations on the chip. These RAMs are called *distributed RAMs*. In terms of the hardware resources required, a distributed RAM is more efficient to implement than an array, because there is no switching between the various RAMs and therefore no need for a multiplexer. However, this efficiency comes at a cost: only one entry of a distributed RAM can be accessed in any one clock cycle, and therefore distributed RAMs are unsuitable in situations where concurrent access to different entries in a RAM is required.

On-chip block RAMs To complement the shallow distributed RAMs that are implemented in CLBs, some FPGAs provide dedicated RAMs known as *block RAMs* or *BRAMs*, which allow for on-chip storage of kilobits of data. Block RAMs are set aside for use as memory, and cannot be used for any other purpose. Therefore, using block RAMs instead of distributed RAMs for on-chip data storage allows one to dedicate more of the configurable hardware resources on the FPGA to the implementation of processing logic. However, using block RAMs has three main disadvantages. First, a block RAM has greater access latency than a distributed RAM, and therefore the use of block RAMs instead of distributed RAMs can result in a slower circuit. Second, because block RAMs have fixed locations, the use of block RAMs reduces the routing

options available to the compiler, and consequently may result in greater routing delays. Third, although a single block RAM holds a large amount of data, the number of block RAMs available on an FPGA is limited. This limits the amount of parallelism across data structures implemented as block RAMs. Block RAMs are suitable in situations where a large amount of data needs to be stored on the chip and where each block RAM is not accessed from many locations on the chip.

Off-chip RAMs In addition to allowing data to be stored on the chip, some FPGAs allow data to be stored in RAMs located off the chip. These RAMs, called *off-chip RAMs*, provide for the storage of megabytes of data. However, there is a higher latency associated with accessing an off-chip RAM than with accessing an on-chip RAM, and off-chip RAMs allow only one read or write access per clock cycle.

3 Research problem

The problem that motivates the research described in this paper is (a) to devise an efficient algorithm for nondeterministic object distribution in P systems, and (b) to develop and evaluate the efficiency of a prototype hardware implementation (based on reconfigurable computing technology) of the devised algorithm.

This research problem fits into our overall research program, which is aimed at the development of a complete hardware implementation of basic cell-like P systems which effectively balances the requirements of performance, flexibility and scalability. In particular, the extent to which the problem can be solved is directly relevant to the question of the feasibility of incorporating an implementation of nondeterministic object distribution into Reconfig-P, our existing hardware implementation of membrane computing.

4 An algorithm for nondeterministic, maximally parallel object distribution in P systems

In this section, we propose an algorithm for nondeterministic, maximally parallel object distribution in P systems.

In order to indicate some of the important issues that were negotiated during the development of the proposed algorithm, and in order to facilitate a comparison of our algorithm with possible alternative algorithms, we present and discuss our algorithm in the context of a wider discussion of the main potential strategies for the solution of the nondeterministic, maximally parallel object distribution problem (henceforth to be referred to as the *object distribution problem*).

4.5 Potential approaches Fundamentally, any effective approach to the object distribution problem must (a) consider a certain space of possible solutions which contains *all* of the solutions to the object distribution problem, (b) be able to correctly distinguish (either explicitly or implicitly) between solutions and non-solutions within this space of possible solutions, (c) randomly select a solution in such a way that all solutions have the same nonzero probability of being selected, and (d) output a solution in a timely manner.

Approaches may differ with respect to the size of the space of possible solutions that is considered. Some approaches may involve an explicit determination of the boundaries of the space of possible solutions to be considered, whereas other approaches may be designed in such a way that an explicit determination is not required.

Approaches may also differ with respect to the way in which they navigate the space of possible solutions. Some approaches are designed in such a way that all non-solutions are avoided, so that only solutions are considered during the navigation of the space of possible solutions. We call such approaches *direct* approaches. Other approaches consider both solutions and non-solutions during the navigation. We call such approaches *indirect* approaches.

To fulfil requirement (c), an approach obviously requires one or more sources of randomness.

4.6 Indirect approaches

Indirect straightforward approach The *indirect straightforward approach* is to simply enumerate all the possible solutions to the object distribution problem that are contained in a certain space of possible solutions (which is known to contain all the solutions to the object distribution problem), and then pick possible solutions at random, checking whether they are actually solutions, until a solution is found.

The indirect straightforward approach considers both non-solutions and solutions as it navigates the space of possible solutions. More specifically, it considers a number of non-solutions before it finds a solution. The algorithm eliminates only one possible solution at a time. The first solution to be found is the solution that is output.

The designers of an algorithm for the solution of the object distribution problem should regard the efficiency of the indirect straightforward approach as a baseline efficiency, and aim for an efficiency significantly higher than this baseline efficiency.

Incremental approach Nondeterministic, maximally parallel distribution of objects to reaction rules is perhaps most intuitively implemented in an incremental manner. In the *incremental approach*, the distribution of objects to a particular reaction rule is accomplished in rounds; that is, a reaction rule may potentially be processed many times. It

is an indirect approach. Martinez, Fernandez, Arroyo and Gutierrez (2007) have proposed an incremental algorithm for nondeterministic, maximally parallel distribution of objects to reaction rules. This algorithm has significantly better efficiency than the indirect straightforward approach. We now describe the algorithm in order to illustrate the fundamental features of the incremental approach.

In the incremental approach, those reaction rules to which it is possible that objects will be distributed are placed in a *pool*. Initially, all of the reaction rules are in the pool. During the course of the object distribution process, reaction rules are removed from the pool. At any given time, the reaction rules in the pool are those that are still under consideration, whereas the reaction rules out of the pool are no longer under consideration. In each iteration, one of the reaction rules in the pool is selected at random, and its number of instances is incremented by a random amount (such that the total number of instances of the reaction rule does not exceed its maximum possible number of instances given the multiplicities of the object types in the region). The multiplicities of the object types in the region are then updated (decreased) according to the number of instances of the reaction rule added in the previous step. At this point the applicability of each of the reaction rules in the pool (given the updated multiplicity values) is checked. If it is impossible for there to be additional instances of a particular reaction rule, then that reaction rule is removed from the pool; otherwise, it remains in the pool. The process repeats until there are no more reaction rules in the pool.

Like the indirect straightforward approach, the incremental approach considers a number of non-solutions before it finds a solution, and outputs the first solution it finds. Unlike the indirect straightforward approach, it may eliminate more than one possible solution at a time (i.e., in a round). As it is impossible for the number of instances of a reaction rule to decrease, at any given time all possible solutions for which the number of instances of any reaction rule is less than the current value for the number of instances of that reaction rule is no longer under consideration. Therefore, in general, the incremental approach converges on a solution more quickly than the indirect straightforward approach.

This approach can be more efficient for many applications. However, the efficiency of the incremental approach depends on the random number of instances that is added to the current number of instances of the selected reaction rule at each iteration. If the random numbers generated are small, the approach can take many iterations to distribute the objects to the reaction rules. This is particularly true when there is a large number of applicable reaction rules in the region and the number of available objects for each object type is large.

It might be more desirable, particularly from the point of view of efficiency, to determine the final number of instances of a reaction rule directly rather than indirectly. In a direct approach, the distribution of objects to a particular reaction rule occurs in one step. To the best of our knowledge, no instances of the direct approach to the implementation of nondeterministic, maximally parallel object distribution have been reported in

the literature. This is a reflection of the general lack of research done so far on implementations of nondeterministic, maximally parallel object distribution, and does not suggest that the direct approach is infeasible. We now describe two direct approaches: the *direct straightforward approach* and our own proposed algorithm.

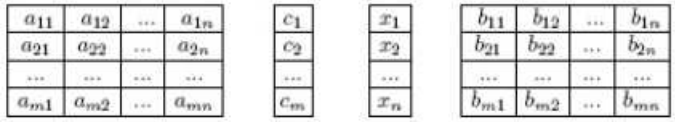
4.7 Direct approaches

Direct straightforward approach In the *direct straightforward approach*, all the solutions to the object distribution problem are given as input, and one of these solutions is simply selected at random. This would be a feasible approach if the multiplicity of each object type in the region were static. In that case, all the possible solutions could be calculated at compile-time, and one of these solutions could be randomly selected at run-time during each transition of the P system. However, given that the multiplicities of the object types in the region change throughout the execution of the P system, the approach is infeasible, mainly because of the large amount of time and hardware resources required for the calculation of all the possible solutions at run-time.

Our proposed approach: the DND algorithm We have devised an algorithm for nondeterministic, maximally parallel object distribution in P systems. Our algorithm, which we call the *Direct Nondeterministic Distribution algorithm* or *DND algorithm*, performs the distribution of objects to a reaction rule in one step (with a possible adjustment step performed before the termination of the algorithm), finds a solution without needing to enumerate possible solutions, and is able to operate without being explicitly aware of many of the characteristics of the space of possible solutions. We now describe how the DND algorithm works.

As shown in Section 2.2, the number of objects available in the region in the current transition and the number of objects for each object type required by each reaction rule can naturally be represented as a linear system of inequalities. If the number of reaction rules is n, then the linear system, if interpreted geometrically, defines an n-dimensional space. This space is the space bounded by the hyperplanes defined by the n inequalities (taken as equations) of the linear system. The possible maximally parallel applications of the reaction rules correspond to certain points within the space. Our approach selects one of these points at random in a direct manner: the random value for each coordinate of the point is determined in one step (plus a possible additional adjustment step in which the value is confirmed or adjusted).

Under the geometric interpretation, each reaction rule corresponds to one of the n dimensions of space. The point closest to the origin at which a hyperplane intersects the axis for a dimension is the least upper bound (*boundary value*) for the number of instances of the reaction rule associated with the dimension. This boundary value is simply the minimum ratio of all the ratios of the number of available objects for an object type and the number of objects of that object type required by the reaction rule.



procedure obtainASolutionNondeterministically (
m: number of object types required by a reaction rule
n: number of reaction rules in the region
A: an $m \times n$ matrix (with initially unmarked columns) used to store the coefficients of the linear system
B: an initially empty $m \times n$ matrix that contains results of calculations carried out on A
C: an $m \times 1$ matrix that contains the RHS constants of the linear system
X: an $n \times 1$ matrix used to store the solution
V: an $m \times 1$ matrix used to store accumulated sums used in the calculation of values to be stored in B
Z: a list of integer labels for columns of A (ordered according to the order in which the columns of A are processed)

Z(k): the k^{th} element of Z

```
    //Forward phase
1.  for u = 1 to n
2.      Randomly select a column p from all the unmarked columns in A (1 ≤ p ≤ n).
3.      Add p to Z.
4.      if B is empty
5.          let q be the minimum value of all c_i/a_ip   (1 ≤ i ≤ m).
6.      else
7.          let q be the minimum value of all b_i(u-1)/a_ip   (1 ≤ i ≤ m).
8.      if p is the only unmarked column in A
9.        if q is an integer
10.           Set x_p = q. End procedure.
11.       else
12.           Set x_p = ⌊q⌋. Go to 22.
13.     else
14.       if q = 0
15.           Set x_p = q and mark x_p as final.
16.       else
17.           Randomly select r ∈ {0, 1, ..., ⌊q⌋} and set x_p = r.
18.           For all i (1 ≤ i ≤ m), set v_i = v_i + r a_ip.
19.           For all i (1 ≤ i ≤ m), set b_iu = c_i − v_i.
20.       Mark column p in A.
21. end for
    //Backward phase
22. Reset V and set v_i = x_Z(n) a_iZ(n) for all i (1 ≤ i ≤ m).
23. for s = n − 1 to 1
24.     if x_Z(s) is not marked as final
25.       if s = 1
26.           let q' be the minimum value of all (c_i − v_i)/ a_iZ(1) (1 ≤ i ≤ m).
27.       else
28.           let q' be the minimum value of all (b_i(s−1) − v_i)/ a_iZ(s) (1 ≤ i ≤ m).
29.       if x_Z(s) ≠ q' set x_Z(s) = ⌊q'⌋ .
30.       For all i (1 ≤ i ≤ m), set v_i = v_i + ⌊q'⌋ a_iZ(s).
31.     end if
32.  end for
```

Fig. 4.3 Pseudocode for the DND algorithm. (For the sake of simplicity of presentation, it is assumed that the coefficient matrix contains only nonzero values.)

At the beginning of the DND algorithm, the linear system is said to have $n - 1$ degrees of freedom. This is because, if $n - 1$ of the dimensions have been assigned a value, the value for the remaining dimension is fixed at a particular value (obviously dependant on the values assigned to the other dimensions). The algorithm starts by selecting a dimension i ($1 \leq i \leq n$) at random. An integer value $v_i \in [0, \beta]$, where β is the boundary value for the dimension, is then selected at random. The value of x_i, the number of instances of the reaction rule associated with dimension i, is set to v_i. The value of x_i is provisional at this stage, unless $x_i = 0$, in which case the value of x_i is final. Even if the current value of x_i is provisional, the algorithm proceeds for the time being on the assumption that the value of x_i is final. That is, the value for dimension i of the random solution is assumed to be determined. (It is this fact that makes the algorithm an instance of the direct approach to the object distribution problem.) The value of x_i is substituted into the linear system, which results in a new linear system with one fewer degree of freedom. This in effect removes the dimension i from consideration and implicitly changes the boundary values for the remaining dimensions. At this point, one of the remaining dimensions — call it dimension j — is selected at random. This dimension is processed in a similar manner to the previously processed dimension (i.e., dimension i): x_j is calculated (and regarded as provisional unless $x_j = 0$), x_j is substituted into the linear system, and a new linear system with one fewer degree of freedom results. The remaining dimensions are processed in a similar manner. When the last dimension is processed — call it dimension k — the degree of freedom is 0. This means that the random value for the dimension (i.e., x_k) has already been determined. Because of the mathematical properties of the object distribution problem, it turns out that x_k is always the floor of the boundary value for the dimension. This value is the final value for x_k, even if it is nonzero. This completes the first phase of the algorithm, which we call the *forward phase*.

At the end of the forward phase, a provisional solution to the object distribution problem has been obtained. Each x_p value ($1 \leq p \leq n$) is the number of instances of the reaction rule associated with dimension p in the provisional solution. It is a key feature of the DND algorithm that the provisional solution obtained during the forward phase is either an actual solution or close to an actual solution that uniquely corresponds to it. If the boundary value calculated for the last dimension to be processed in the forward phase (i.e., dimension k) is an integer, and each reaction rule requires at least one object of each object type (so that the relevant coefficients in the linear system are all nonzero), then the provisional solution is an actual solution. In this case, no further computation is required, and the algorithm terminates. Otherwise, a phase of the algorithm called the *backward phase* commences. In the backward phase, the provisional solution that was obtained during the forward phase is adjusted (if necessary) so that it coincides with its corresponding actual solution.

In the backward phase, the dimensions are processed in reverse order (with respect to the order randomly chosen in the forward phase). Dimension k, the last dimension to be processed in the forward phase, is skipped, because the value for x_k is already final.

In processing a dimension q ($q \neq k$), the x_r values ($r \neq q$) for the other dimensions (regardless of whether they are provisional or final) are regarded as fixed (i.e., these values are substituted as is into the linear system), and m_q — the maximum value of x_q (given the inequality associated with dimension q in the linear system) — is determined. The value of x_q is either confirmed (if it is the same as m_q) or adjusted to m_q. When the final x_p values ($1 \leq p \leq n$) for all dimensions have been determined, the final solution to the object distribution problem has been obtained, and the algorithm terminates.

Unlike algorithms based on the incremental approach, the DND algorithm does not need to navigate through a series of non-solutions. It is designed in such a way that it always converges on a solution without needing to process non-solutions. In fact, it avoids the processing of non-solutions, and exhibits no bias towards any solution, without needing to perform any explicit reasoning about the space of possible solutions as a whole.

Pseudocode for the DND algorithm is shown in Figure 4.3.

4.8 Explanation of our proposed approach In this section, we attempt to provide some intuition for the principles behind the operation of the DND algorithm. Again we exploit the analogy with n-dimensional geometry.

To allow a graphical presentation, we project all the solutions to the object distribution problem (i.e., all the possible maximally parallel applications of the reaction rules) in n-dimensional space onto two-dimensional space using parallel coordinates. Parallel coordinates are often used in the visualisation and analysis of data in $n > 3$ dimensions. If one regards an n-dimensional datum as a point in n-dimensional space, then the method of parallel coordinates can be explained as follows. To show a set of points in n dimensions, a backdrop consisting of n vertical and equally spaced parallel axes is drawn. A point is represented as a polyline with vertices on the parallel axes, where the position of the vertex on the i^{th} axis corresponds to the i^{th} coordinate of the point.

We use parallel coordinates in the following way. Each parallel axis corresponds to one of the x_i ($1 \leq i \leq n$) (i.e., the number of instances of the reaction rule associated with dimension i). There are n reaction rules, and so there are n parallel axes.

A solution to the object distribution problem corresponds to a polyline with exactly one vertex on each parallel axis, where the position of this vertex corresponds to the number of instances calculated for the reaction rule associated with the axis.

In Figures 4.4 and 4.5 we present two examples to show in broad outline how the DND algorithm works. The P system used in the examples is shown in Figure 2.1. Thus we have $n = 4$. In the examples, we suppose that the random order in which the reaction rules are to be processed is r_4, r_2, r_3, r_1. All of the solutions to the object distribution problem for the current transition of the example P system are shown in Figure 2.1.

Example 1

Initially, the linear system has $n-1$ (= 3) degrees of freedom. Since in the DND algorithm, each reaction rule is processed in only one step, the algorithm determines the solution (in the forward phase) in n (= 4) steps. The algorithm starts by selecting a reaction rule at random. According to our assumption, the reaction rule that is chosen first is r_4. The algorithm determines the maximum possible value (boundary value) for x_4 (i.e., the number of instances of r_4). The boundary value is determined to be 3, so the range of values on the parallel axis associated with x_4 is [0, 3]. The algorithm then selects an integer value in this range at random. Suppose that the value it selects is 0. This means that the value of x_4 is set to 0. This provisional value of x_4 will either be equal to the final value of x_4 (if the algorithm 'guessed' correctly) or smaller than the final value of x_4 (if the algorithm guessed incorrectly). Consequently, all values smaller than the provisional value are filtered out. Nevertheless, as there are no solutions to the object distribution problem for which $x_4 < 0$, no solutions are eliminated, and so all of the solutions still have a chance of being output by the algorithm (see Figure 6a). The provisional value of x_4 is substituted into the linear system, which means that the algorithm now proceeds on the assumption that $x_4 = 0$. In effect r_4 is no longer under consideration, and so the updated linear system now has 2 instead of 3 degrees of freedom. At this point, the algorithm selects another reaction rule at random. By our assumption, r_2 is selected. When processing r_2 (see Figure 4.6b), the algorithm determines that the updated boundary value for x_2 is 2, so the range of values on the parallel axis associated with x_2 is [0, 2]. Suppose that the algorithm selects 1 as the provisional value for x_2. This eliminates all the solutions to the object distribution problem for which $x_2 < 1$. The polylines associated with the eliminated solutions are now shown in grey on the graph. The value of x_2 is then substituted into the linear system, effectively removing r_2 from consideration, and resulting in a new linear system with 1 degree of freedom. At this stage, the algorithm randomly selects the reaction rule r_3. When processing r_3 (see Figure 4.6c), the algorithm determines that the updated boundary value for x_3 is 4. Therefore the range of values on the parallel axis associated with r_3 is [0, 4]. Suppose that the algorithm selects 1 as the provisional value for x_3. As a result, all solutions to the object distribution problem for which $x_3 < 1$ (that have not yet been eliminated) are eliminated. The polylines associated with these eliminated solutions are now shown in grey. The provisional value for x_3 is substituted into the linear system, effectively removing r_3 from consideration, and resulting in a new linear system with 0 degrees of freedom. The algorithm now moves on to process r_1, the last remaining reaction rule. As the degree of freedom now is 0, x_1 is set to be the updated boundary value for x_1, which is 1. This eliminates all the possible solutions for which $x_1 < 1$. Since the boundary value is an integer, no application of the floor function is required, no backward phase is required, and the algorithm terminates.

Thus the nondeterministically chosen solution to the object distribution problem is: $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, $x_4 = 0$. This solution corresponds to the only remaining polyline in Figure 4.6d.

Fig. 4.4 An example of the operation of the DND algorithm, in which only the forward phase is executed.

5 Evaluation of the DND algorithm

In this section, we evaluate the correctness and theoretical time and space complexity of the DND algorithm.

5.9 Time complexity We now compare the time complexity of the DND algorithm with the time complexity of the incremental algorithm described in Section 4.6. In our time complexity analysis, we make a number of simplifying assumptions. First, the comparison is done at the conceptual/algorithmic level rather than at the implementation level. Second, we do not take into account relatively insignificant operations performed by the algorithms. Third, we assume, for both algorithms, that certain operations have the same time complexity in all situations, even though in reality this time complexity

Example 2

In Example 1 (see Figure 4.4), it was not necessary for the DND algorithm to execute the backward phase. We now alter the example slightly in order to illustrate the backward phase.

Suppose that in Example 1 the algorithm had set the value of x_3 to 3 instead of to 1. In this case, all solutions to the object distribution problem that had not yet been eliminated for which $x_3 < 3$ would have been eliminated. However, since there is no solution to the object distribution problem for which $x_3 = 3$, a new polyline going through $x_3 = 3$ needs to be introduced (see the dotted polyline in Figure 4.6e). With the value of x_3 set to 3, the boundary value calculated for x_1 is 0.5 (see Figure 4.6f). Since the value for each x_i ($1 \leq i \leq n$) must be an integer, the algorithm performs the floor function on 0.5 and therefore sets the value of x_1 to 0 (see Figure 4.6g). The fact that the boundary value for the last processed x_i value (i.e., x_1) was not an integer alerts the algorithm to the fact that it needs to execute the backward phase. In the backward phase, the algorithm processes each of the x_i values in reverse (with respect to the order followed in the forward phase), with the exception of the last x_i value to have been processed in the forward phase, which is skipped. In this example, the algorithm skips x_1 (leaving its value set to 0), processes x_3 (increasing its value to 4), processes x_2 (leaving its value at 1), and then finally processes x_4 (leaving its value at 0). When processing an x_i value, the algorithm calculates the maximum possible value for the x_i value on the supposition that the other x_i values are fixed at their current (not necessarily final) values. For example, when processing the x_3 value, it is assumed that $x_4 = 0$, $x_2 = 1$ and $x_1 = 0$. Given these constraints, the maximum possible value for x_3 is 4, so the algorithm sets the value of x_3 to 4.

The solution that is output by the algorithm in this example corresponds to the polyline shown in Figure 4.6h. Thus the nondeterministically chosen solution to the object distribution problem is: $x_1 = 0$, $x_2 = 1$, $x_3 = 4$, $x_4 = 0$.

Fig. 4.5 An example of the operation of the DND algorithm, in which both the forward phase and backward phase are executed.

may vary depending on certain characteristics of the input. For example, the time taken to find the boundary value associated with a reaction rule depends on the specific definition of the reaction rule, but in our analysis we do not take into account the specifics of this definition, and so assume a fixed time complexity for the operation. Finally, we assume that all the reaction rules in the region are applicable in the current transition.

We consider the performance of the algorithms in the best-case, average-case and worst-case scenarios. The best-case scenario is the situation in which it so happens that as many objects as possible are distributed to the first reaction rule to be selected, and there are not enough objects remaining for any objects to be distributed to any of the other reaction rules. The average-case scenario is the situation in which all reaction rules are processed[10]. The worst-case scenario is the situation in which all relevant factors that cause an increase in time complexity are in effect.

The results of the time complexity analysis are shown in Table 5.1. We now comment on the time complexity results.

In the best case, the two algorithms have similar time complexities. However, the DND algorithm devotes more time than the incremental algorithm to the random selection of reaction rules. The incremental algorithm is able to detect, straight after distributing

[10]Note that a reaction rule may be processed without being assigned any objects.

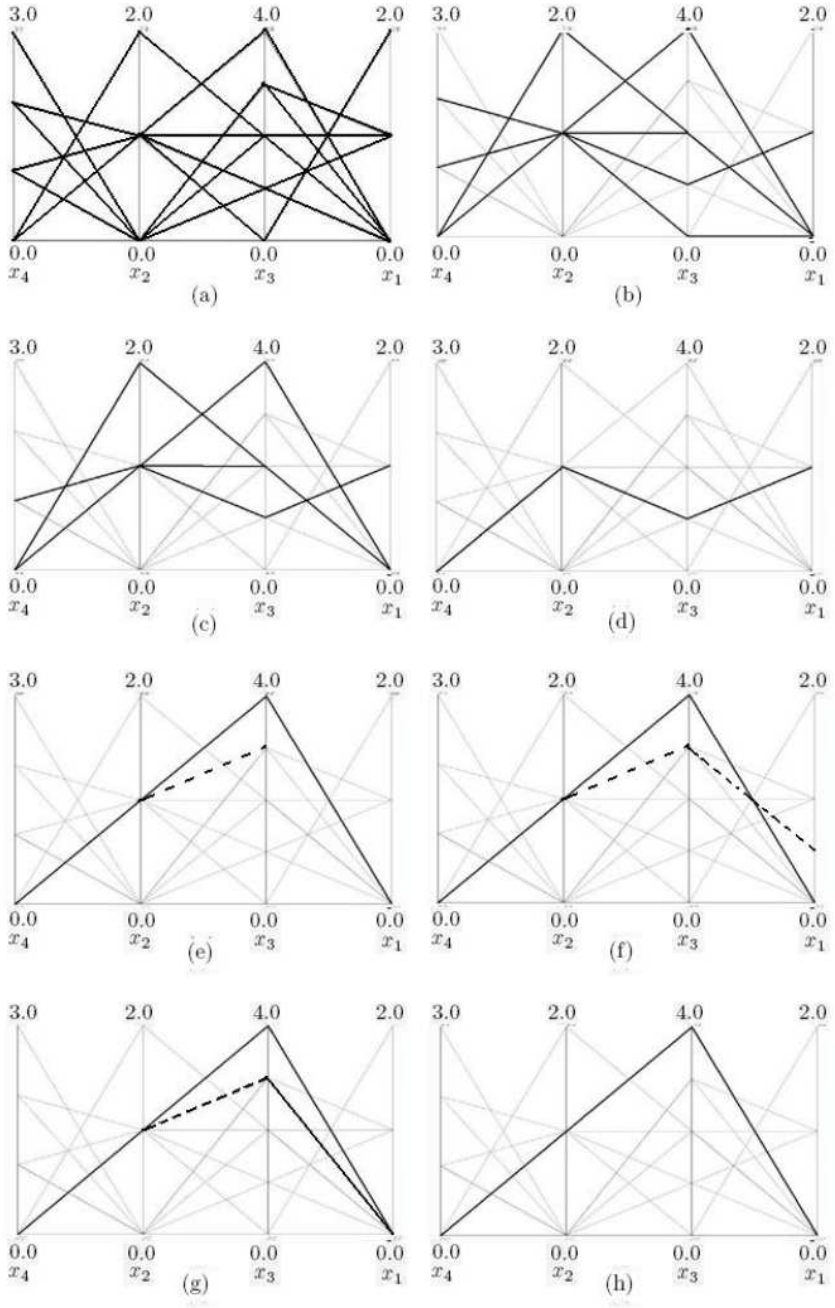


Fig. 4.6 Graphs used in the explanation of the examples presented in Figures 4.4 and 4.5.

Definitions

n is the number of reaction rules in the region.

t_{sr} is the time taken to randomly select a reaction rule from a collection of reaction rules whose size is unknown at compile-time.

t'_{sr} is the time taken to randomly select a reaction rule from a collection of reaction rules whose size is known at compile-time.

t_{bv} is the time taken to calculate the boundary value for the number of instances of a reaction rule.

t_{rv} is the time taken to select a value from a range of valid values at random and assign this value to a reaction rule as its number of instances.

t_u is the time taken to update the multiplicities of available object types in the region.

t_{cv} is the time taken to check whether the current value for the number of instances of a reaction rule is final.

p_i is the number of times reaction rule r_i is processed.

q_i^j is the number of reaction rules still under consideration when reaction rule i is being processed in the j^{th} iteration of the algorithm (a positive integer).

In the table below, we abbreviate $t_{bv} + t_{rv} + t_u$ (i.e., the time taken to process a reaction rule) as t_{pr}.

Time complexities

	Best case	Average case	Worst case
DND algorithm	$nt'_{sr} + t_{pr} + nt_{cv}$	Between $n(t'_{sr} + t_{pr})$ and $n(t'_{sr} + t_{pr} + t_u + t_{cv})$	$n(t'_{sr} + t_{pr} + t_u + t_{cv})$
Incremental algorithm	$t_{sr} + t_{pr} + nt_{cv}$	$\sum_{i=1}^{n} p_i(t_{sr} + t_{pr}$ $+ \sum_{j=1}^{p_i} q_i^j t_{cv})$, where p_i, q_i are relatively small	$\sum_{i=1}^{n} p_i(t_{sr} + t_{pr}$ $+ \sum_{j=1}^{p_i} q_i^j t_{cv})$, where p_i, q_i are large

Table 5.1 Time complexity analysis for the DND algorithm and incremental algorithm.

objects to the first selected reaction rule, that none of the remaining reaction rules is able to obtain any of the remaining objects by checking in a *pre-defined* order the applicability of each reaction rule given the updated multiplicities of objects in the region, and so does not need to make any further random selections of reaction rules. The DND algorithm, on the other hand, checks the remaining reaction rules in a *random* order, and therefore must perform one random selection per remaining reaction rule.

In the average case, in the DND algorithm, every reaction rule is processed at least once, and possibly some of the reaction rules are subjected to an additional checking process, during which it is determined whether the number of instances of the reaction rule

should be adjusted and, if so, by how much. In the incremental algorithm, each reaction rule is processed p_i times and is subject to $\sum_{j=1}^{p_i} q_i^j$ checking processes, during each of which it is determined whether the number of instances of the reaction rule can be increased. The values for p_i and q_i are in general of a moderate magnitude.

In the worst case, in the DND algorithm, each reaction rule is processed once and also subjected to one checking process. In the incremental algorithm, the time complexity is the same as in the average case, but the values for the constants p_i and q_i are in general much larger.

The above observations suggest that the performance of the DND algorithm, at least in terms of theoretical time complexity, is similar to that of the incremental algorithm in the best case, clearly surpasses that of the incremental algorithm in the average case, and significantly surpasses that of the incremental algorithm in the worst case. For the average and worst cases, the time complexity of the DND algorithm depends on n, whereas the time complexity of the incremental algorithm depends on n, each p_i and each q_i ($1 \leq i \leq n$). In the average case, if the p_i and q_i values are small, although the DND algorithm performs better than the incremental algorithm, the difference in performance between the algorithms is not large (especially if n is small). However, as the p_i and q_i values increase, the performance of the incremental algorithm diminishes in comparison with the performance of the DND algorithm. In the worst case, the p_i and q_i values are very large, and the performance of the DND algorithm is much better than that of the incremental algorithm.

5.10 Space complexity Let m be the number of object types in the region and n the number of reaction rules in the region. Both the DND algorithm and incremental algorithm need to store (a) for each reaction rule in the region, the number of objects of each object type required for the application of one instance of the reaction rule, and (b) for each object type in the region, the current multiplicity of the object type. The space complexity for these two items in both algorithms is $O(mn + m)$. The DND algorithm, unlike the incremental algorithm, requires an additional matrix, called the *traceback matrix* (see Section 6.12), for the storage of data related to the updating of the linear system. The traceback matrix contains mn elements. Therefore the overall space complexity for the DND algorithm is $O(2mn + m)$, whereas the overall space complexity for the incremental algorithm is $O(mn+m)$. Thus, at least at the algorithmic level, the incremental algorithm is more efficient in terms of space consumption than the DND algorithm.

5.11 Evaluation of the correctness of the DND algorithm A software program written in Java has been developed in order to empirically verify the correctness of the DND algorithm. This program is able to (a) create linear systems with random numbers of rows and columns and random values for the coefficients and RHS constants, (b) generate all solutions to the object distribution problem for a given linear system,

(c) generate a random solution to the object distribution problem for a given linear system using the DND algorithm, and verify that this solution is indeed a solution, and (d) record the sequence of random solutions to the object distribution problem for a given linear system, determine the frequency of occurrence of each random solution, and compare the set of random solutions obtained with the set of all solutions.

Verification of the correctness of the results produced by the DND algorithm We have empirically tested, using the Java program mentioned above, the correctness of the results produced by the DND algorithm. One million different randomly generated linear systems were used as input during the testing. Both the number of rows and the number of columns in a matrix in the linear system were limited to 20, and the value of each coefficient and RHS constant was limited to 20. It was found that every solution output by the DND algorithm during the testing was correct.

Verification of the ability of the DND algorithm to cover all solutions Since it might appear that the DND algorithm takes a 'short cut' when finding a solution to the object distribution problem, it is important to verify that (a) the algorithm is able to generate all the solutions in the solution space for an instance of the object distribution problem, and (b) the algorithm does not have any positive or negative bias towards any solution. To verify these two properties of the algorithm, we have performed statistical analyses of the results produced by the algorithm for various input linear systems. Figure 5.7b illustrates the coverage exhibited by the algorithm for an example input linear system, for which there are 19 possible solutions, and where the algorithm was executed approximately two million times.

Verification of the sufficient randomness of the sequence of solutions output by the DND algorithm The solutions output by the DND algorithm must be produced in a sufficiently random manner. To investigate the ability of the algorithm to produce solutions in a sufficiently random manner, it is necessary to analyse the sequence of solutions it generates when executed many times. Figure 5.7a shows the sequence of solutions that were output by the algorithm, and the frequency of occurrence of each solution, for a particular experiment. In this experiment, there were 56 solutions and the algorithm was executed 560 times. The results of the experiment suggest that the DND algorithm produces solutions in a sufficiently random manner.

6 Description of a hardware implementation of the DND algorithm

In this section, we describe a prototype hardware implementation of the DND algorithm that uses the reconfigurable computing technology outlined in Section 2.4. For the prototype, we used a Xilinx Virtex II FPGA. First we describe the major data structures and

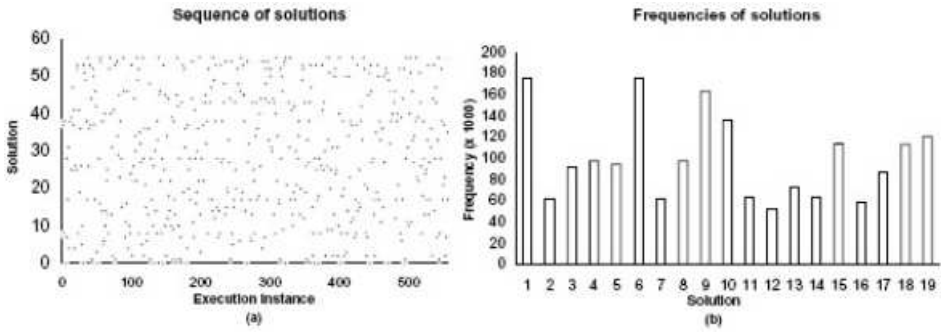


Fig. 5.7 (a) The sequence of solutions output by the DND algorithm, and (b) the frequency of occurrence of each solution, for a particular experiment.

processing units in the implementation, and show how they interact during the execution of the DND algorithm. Then we discuss the degree to which parallelism is achieved in the implementation. Finally, we discuss optimisations that we implemented in order to increase the efficiency of the implementation.

In order to reduce the hardware resource consumption and increase the execution time performance of an implementation based on reconfigurable hardware, attempts are commonly made to (a) tailor the design of the hardware components to the special characteristics of the input, (b) minimise communication between hardware components, and (c) use constants instead of variables where possible (since constants, but not variables, can be hard-coded into the circuit). We employed these strategies during the development of Reconfig-P, and the result was a substantial benefit in terms of hardware resource consumption and execution time performance. However, the effectiveness of these strategies is limited when they are applied to the implementation of nondeterministic object distribution. As nondeterministic object distribution is an inherently uncertain process, multiple alternative scenarios need to be accommodated. This places constraints on the use of constants and hard-coded logic, and therefore limits the ability to optimise the hardware circuit that is generated for a particular input application. In addition, the introduction of nondeterminism inevitably increases the amount of communication between hardware components, mainly because of the more complex interplay between reaction rules. Thus implementing nondeterministic object distribution is more challenging than implementing deterministic object distribution. Our purpose in developing a prototype hardware implementation of the DND algorithm was to investigate the feasibility of producing an efficient implementation of nondeterministic object distribution.

6.12 Data structures The major data structures in our prototype implementation are: (a) the two-dimensional coefficient matrix for the linear system, (b) a two-dimensional *traceback matrix* that records intermediate results obtained during the forward phase of the algorithm, (c) the one-dimensional matrix containing the RHS constants of the linear system, (d) an array for the storage of accumulated sums that are calculated during the

execution of the algorithm, and (e) an array for the storage of the solution to the object distribution problem that is output by the algorithm. We describe the implementation of these data structures below.

Coefficient matrix The coefficient matrix (called `CoefficientMatrix`) corresponds to the matrix *A* in the pseudocode for the DND algorithm in Figure 4.3. Hence it is a two-dimensional matrix with m rows and n columns, where m is the number of object types in the region and n is the number of reaction rules in the region.

Although our prototype implementation of the DND algorithm does not aim to support parallelism across reaction rules, we leave open the possibility of implementing some degree of parallelism in a future implementation by using registers rather than RAMs in the implementation of `CoefficientMatrix`. In our implementation, each row of `CoefficientMatrix` is implemented as an array of read-only registers called `CoefficientMatrixRow`. This allows parallelism both across the rows (corresponding to the object types) and across the columns (corresponding to the reaction rules) of the matrix. However, this high degree of potential parallelism comes at the cost of a more complicated hardware circuit when the implemented array is large (i.e., when there is a large number of reaction rules in the region).

Traceback matrix The traceback matrix (called `TracebackMatrix`) corresponds to the matrix *B* in the pseudocode for the DND algorithm in Figure 4.3. Each column in this matrix records intermediate results obtained during the forward phase of the algorithm. Only one column of the matrix is accessed at a time. Since concurrent access to the elements in a matrix row is not required, each matrix row is implemented as an n-entry p-bit distributed RAM called `TracebackMatrixRow`, where p is the bitwidth of the coefficient values stored in the matrix. With each matrix row implemented as a separate RAM, all elements in a column can be accessed concurrently.

It might be thought that, if the coefficient matrix has a large number of columns (i.e., if there is a large number of reaction rules in the region), it would be better to implement the matrix using block RAMs instead of distributed RAMs, because in this way hardware resources (specifically, LUTs) could be saved for other purposes. The Xilinx II FPGA used in the implementation provides up to 3 MB of dedicated on-chip memory, organised into 144 18Kb block RAMs. As the default bitwidth for a coefficient value is 8 bits, each block RAM on the FPGA has a depth of approximately 2 Kb, which is enough to meet the storage requirements for a large number of columns. However, given the disadvantages of block RAMs identified in Section 2.4, using block RAMs is most suitable when the number of reaction rules per region is large and the number of object types per region and/or the number of regions is small. If these conditions are not satisfied, then it is probably more efficient to use distributed RAMs.

In our implementation, by default, `TracebackMatrix` is implemented using distributed RAMs. However, depending on the specific characteristics of the input application, the

user can change the default setting in order to configure the implementation to use block RAMs only or to use both block RAMs and distributed RAMs.

Other matrices Matrices *C*, *V* and *X* in the pseudocode for the DND algorithm in Figure 4.3 are implemented as `RHSConstantsArray`, `AccumulatedArray` and `Solution Array`, respectively. Each of these matrices is implemented as an array of registers to allow its elements to be accessed in parallel.

6.13 Processing units The major processing units in the prototype implementation of the DND algorithm include a random number generator and processing units that implement the logic of the DND algorithm. We describe the implementation of these processing units below.

Random number generator A random number generator (called `RandomNumber Generator`) is included in the implementation in order to realise (as closely as reasonably practicable) the nondeterminism of the DND algorithm.

A variety of methods of generating random numbers (really, pseudorandom numbers) have been studied. Most of these methods involve the use of arithmetical functions. However, because the implementation of arithmetical functions can generate deep logic when a language such as Handel-C is used, many of these methods are not suitable for adoption in our implementation.

A random number generator that is commonly implemented on FPGAs is the Linear Feedback Shift Register (LFSR). An LFSR is a shift register whose input is the result of performing the XOR (or XNOR) operation on certain bits of the shift register in its previous state. Initially, the bits in an LFSR are initialised in a random manner[11]. Then a right-shift operation is performed. An XOR (or XNOR) operation is performed on certain bits in the shift register, and then the result is put into the left-most bit of the shift register. This bit may be regarded as the random number generated by the LFSR. Then the process iterates. Although it reduces the quality of the random numbers generated somewhat, it is possible to construct a random number by stringing together n distinct bits from the LFSR in one of its states, rather than by using n LFSRs and stringing together the random bits from each of the LFSRs. As the XOR/XNOR operation is very efficient to implement on an FPGA, LFSRs are among the most efficient random number generators for FPGAs. Figure 6.8 shows an example 32-bit LFSR.

Implementing a 32-bit shift register in a standard way using flip-flops (registers) consumes approximately 16 slices (4 CLBs) since there are two flip-flops per slice (see Figure 2.2). However, the Virtex II FPGA used in the implementation provides a particularly efficient means of implementing LFSRs. It provides a macro that can configure

[11] The ultimate source of this randomness is an external random number generator used by the software program that generates the Handel-C code for the implementation.

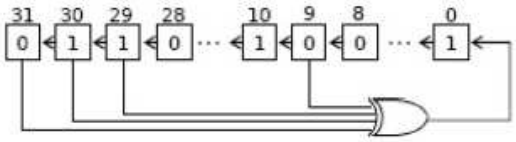


Fig. 6.8 A 32-bit Linear Feedback Shift Register (LFSR). In this LFSR, bits 9, 29, 30 and 31 are the only bits to which the logical operation (in this case the XOR operation) is applied.

an LUT into a 16-bit shift register, thereby enabling the implementation of a 32-bit shift register using only two LUTs (i.e., approximately one slice).

In our implementation, since the object distribution processes for different regions occur in parallel, a 32-bit LFSR was included for each region. These LFSRs execute in parallel with the other processing units, and constantly produce random numbers. Since the default bitwidth for data types in the implementation is 8 bits, the last 8 bits of the LFSR are regarded as representing the random number generated by the LFSR.

The DND algorithm requires random numbers within a certain dynamically determined range to be generated. Although an LFSR can generate random numbers with a certain number of bits, it is unable by itself to generate random numbers within a dynamically determined range. In our implementation, a random number within the desired range is obtained by dropping one bit every clock cycle (starting from the most significant bit) from the original 8-bit random number until the random number is within the required range.

Processing units that implement the logic of the DND algorithm As described in Section 4.7, the DND algorithm proceeds in two phases: a forward phase and a backward phase. Similar operations are performed in the forward phase and backward phase. To prevent the Handel-C compiler from generating redundant hardware components, the operations that are expensive in terms of hardware resources are implemented as separate modules which can be invoked by the components that implement the main procedures for the forward phase and backward phase. These operations are `FindBoundaryValue`, `SubstituteValue` and `UpdateRHSConstants`. `FindBoundaryValue` calculates the ratios between the RHS constants and the respective coefficient values of a variable, and returns the floor of the minimum ratio as the boundary value for the variable. `SubstituteValue` computes for each coefficient value associated with a variable the product of the coefficient value and the value of the variable, and updates `AccumulatedArray` by adding each of the products to the value currently stored in the appropriate element of `AccumulatedArray`. `UpdateRHSConstants` calculates for each RHS constant value the difference between the RHS constant value and the corresponding value stored in `AccumulatedArray`, and stores these differences in the column of

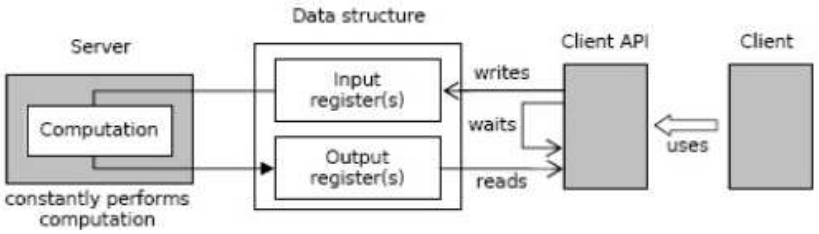


Fig. 6.9 An illustration of a hardware module implemented according to the client-server architecture.

`TracebackMatrix` that corresponds to the variable currently being processed.

The `FindBoundaryValue`, `SubstituteValue` and `UpdateRHSConstants` processing units are each implemented using a client-server architecture (see Figure 6.9). At the centre of the client-server architecture is a data structure that consists of input registers and output registers. A server processing unit is responsible for performing a computation using the data in the input registers and storing the output of the computation in the output registers. It performs the computation constantly. A client processing unit that wishes to use the server uses a client API to feed new input into the input registers when it requires the computation to be performed, and to read the result of the computation from the output registers (after waiting for the appropriate number of clock cycles).

Figure 6.10 shows the Handel-C code for a specific instance of the `SubstituteValue` operation.

For each region of the P system, the implementation includes — in addition to the `FindBoundaryValue`, `SubstituteValue` and `UpdateRHSConstants` processing units for the region — two processing units called `ForwardPhaseProcessor` and `BackwardPhaseProcessor`.

`ForwardPhaseProcessor` and `BackwardPhaseProcessor` are clients to the `FindBoundaryValue`, `SubstituteValue` and `UpdateRHSConstants` processing units. `ForwardPhaseProcessor` implements the main procedure of the forward phase of the DND algorithm. When processing a reaction rule, it invokes the aforementioned processing units, and also performs its own operations, such as assigning a random value to the reaction rule and writing intermediate results into `TracebackMatrix`. `BackwardPhaseProcessor` implements the main procedure of the backward phase of the DND algorithm. It operates in a similar manner to `ForwardPhaseProcessor`, but processes the reaction rules in reverse order. When processing a reaction rule, `BackwardPhaseProcessor` either adjusts or confirms the value that was assigned to the reaction rule in the forward phase.

```
struct _SubstituteValueStructure
{
  unsigned int 8 A0, A1;
  unsigned int 8 B ;
  unsigned int 8 C0, C1;
}
typedef struct _SubstituteValueStructure
SVStruct;
macro proc SubstituteValueServer(SVPtr)
{
  /* Constantly performs the computation
*/
  while(1)
  {
   par
   {
    SVPtr->C0= SVPtr->B*SVPtr->A0;
    SVPtr->C1= SVPtr->B*SVPtr->A1;
   }
}
```

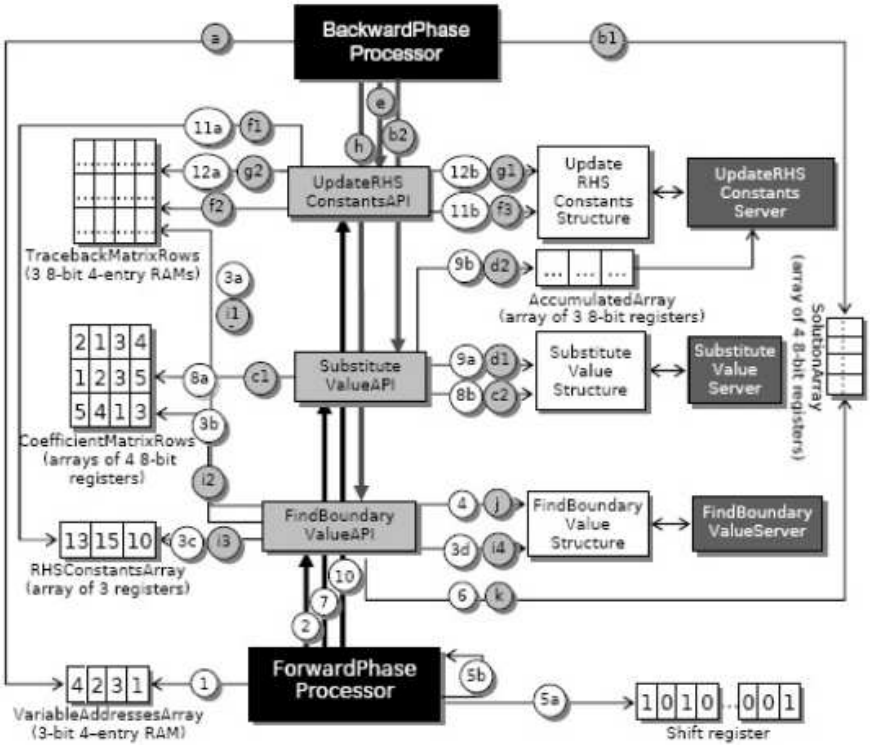
```
macro proc SubstituteValueAPI
(SVPtr, Address, Value)
{
  /* Send data to server */
  par
  {
   SVPtr->A0=CoefficientMatrixRow0[Address];
   SVPtr->A1=CoefficientMatrixRow1[Address];
   SVPtr->B=Value;
  }
  /* Wait for substitution to be
performed */
  delay;
  /* Receive (and store) data from
server */
  par
  {
  AccumulatedArray[0]+=SVPtr->C0;
  AccumulatedArray[1]+=SVPtr->C1;
  }
}
```

Fig. 6.10 An example of Handel-C code for the `SubstituteValue` operation, an operation which uses the client-server architecture.

6.14 Interaction between the data structures and processing units Figure 11 shows a high-level view of the interactions that occur between the various data structures and processing units during the execution of the DND algorithm.

6.15 Parallelism The presentation of the DND algorithm in Figure 4.3 is intended to indicate the fundamental nature of our proposed approach to the solution of the object distribution problem. It does not take into account characteristics of the specific computing platform on which the algorithm is to be executed. However, because any consideration of the implementation of the DND algorithm on a hardware-based parallel computing platform should include an investigation of possible ways of implementing the algorithm as a parallel algorithm in order to minimise its time complexity, we now indicate some of the ways in which the DND algorithm can be implemented in a parallel manner.

Regarding system-level parallelism, it is obvious that the instances of the DND algorithm for the different regions of the P system can be executed in parallel. Regarding region-level parallelism, during the processing of a reaction rule, operations that are performed on individual object types associated with the reaction rule (i.e., performed on rows of the relevant matrices) can be executed in parallel. That is, parallelism across object types can be achieved. However, because of the interdependence between the numbers of instances of the reaction rules to be processed by the DND algorithm, there is no potential for the exploitation of parallelism across the main processing performed



Forward phase

1: obtain address for current variable from
2: pass address of current variable to
3a: obtain intermediate RHS constants for current variable from
3b: obtain coefficients for current variable from
3c: obtain RHS constants for current variable from
3d: pass RHS constants (or intermediate RHS constants) and coefficients to
4: obtain boundary value for current variable from
5a: obtain random number from
5b: process random number to make it less than or obtain RHS constants from
6: store random provisional value or final value for variable in
7: pass random provisional value for current variable to
8a: obtain coefficients for current variable from
8b: pass coefficients to
9a: obtain products of variable value with coefficients from
9b: store products in
10: pass address of current variable to
11a: obtain RHS constants for current variable from
11b: pass RHS constants to
12a: obtain intermediate RHS constants from
12b: store intermediate RHS constants in

Backward phase

a: obtain addresses for previous, current and next variables from
b1: obtain final value of previous variable from
b2: pass final value to
c1: obtain coefficients for previous variable from
c2: pass final value and coefficients to
d1: obtain products of final value with coefficients from
d2: store products in
e: pass addresses of current and next variables to
f1: obtain RHS constants from
f2: obtain intermediate RHS constants for next variable from
f3: pass RHS constants or intermediate RHS constants to
g1: obtain updated intermediate RHS constants for current variable from
g2: store intermediate RHS constants in
h: pass address of current variable to
i1: obtain coefficients for current variable from
i2: obtain intermediate RHS constants for current variable from
i3: obtain RHS constants for current variable from
i4: pass RHS constants (or intermediate RHS constants) and coefficients to
j: obtain boundary value for current variable from
k: store boundary value for current variable in

Fig. 6.11 A high-level view of the interactions that occur between the various data structures and processing units in the prototype implementation during the execution of the DND algorithm.

for these reaction rules (i.e., across columns of the relevant matrices)[12]. Even so, there are some operations that can be carried out in parallel in order to speed up the DND algorithm (in both the forward phase and backward phase). For instance, lines 5, 14 and 15 in the pseudocode in Figure 4.3 can be executed for all the reaction rules before the start of the DND algorithm so that those reaction rules to which it is impossible that objects be distributed can be entirely removed from consideration immediately. In this way, the DND algorithm would need only to process a subset of the reaction rules. In addition, during the execution of the DND algorithm, lines 7, 14, 15 and 28 can be concurrently and constantly executed for all the reaction rules other than the reaction rule currently being processed in the usual way by the DND algorithm. This would result in the numbers of instances of some of the reaction rules being preemptively set to zero or finalised. These reaction rules would not need to be processed by the DND algorithm in the usual way. This would enable the detection of the following two types of situations: (a) at the end of the forward phase, no values need to be adjusted, and (b) at some point during the backward phase, none of the remaining values needs to be adjusted. In the first case, the backward phase can be skipped entirely, even if the boundary value for the last reaction rule to be processed in the forward phase is not an integer. In the second case, it would be possible to preemptively terminate the backward phase.

The purpose of our existing implementation of the DND algorithm is to investigate the feasibility of incorporating an efficient implementation of nondeterministic object distribution into Reconfig-P. So in this work we are interested primarily in the average-case execution of the DND algorithm. In the average case, all of the reaction rules need to be processed in the usual manner, one after the other (i.e., it is not possible to preemptively set the number of instances of any of the reaction rules to zero). Thus, our current implementation of the DND algorithm achieves full parallelism at the system level and parallelism across object types at the region level. If we determine that it is feasible to incorporate an efficient implementation of the DND algorithm into Reconfig-P, we will take advantage of the existing parallelism across reaction rules achieved by the current version of Reconfig-P in the implementation of the parallelised preemptive termination strategies outlined above. At present, `CoefficientMatrix` is implemented as an array of registers to make the realisation of these strategies possible.

Given that our prototype implementation achieves system-level parallelism, the number of clock cycles taken to complete the object distribution process for every region is equal to the number of clock cycles taken to complete the most time-consuming of these processes. Within each region, the major processing units that perform computations across object types (such as `SubstituteVariable-Server` and `UpdateSystemServer`) take only one clock cycle to complete their respective computations. Also performing its computation across object types, the `Find BoundaryValueServer` processing unit, provided that the standard Handel-C imple-

[12] Although it is conceivable that one could develop a nondeterministic algorithm that implements parallelism across the main processing for the reaction rules, such an algorithm is likely to involve complicated and potentially numerous rollbacks.

mentation of division is used, takes $\log_2 m$ clock cycles to complete its operations, where m is the number of object types.

6.16 Optimisations aimed at reducing path delays in the circuit Typically the execution of an operation on a digital circuit involves a *routing delay* (because of the time taken for the transmission of inputs and outputs) and a *logic delay* (because of the time taken for data to be passed through a series of logic gates). Roughly, the *path delay* for an operation is the sum of the routing delay and the logic delay associated with the execution of that operation. As the clock rate of the whole hardware system is determined by the longest path delay on the circuit, it is important to evenly distribute the path delays among the various operations.

Instead of manually placing hardware components at specific locations on the chip, it is possible in our case to reduce the path delay in an indirect manner by generating the Handel-C code in such a way that it can be converted by the Handel-C compiler and Xilinx synthesis tools into a circuit with efficient placement and routing.

We now briefly describe two methods by which we reduced path delays in our implementation: duplication of hardware components and logic depth reduction.

Duplication of hardware components In the implementation of an operation, wherever possible, if the cost of routing data to/from a component is greater than the cost of implementing a duplicate component at a more optimal location, then a duplicate component is implemented. For example, in the forward and backward phases of the DND algorithm, different arrays of column indices are used, even though in principle a single array of column indices could be used for both phases.

Logic depth reduction In our implementation, since an operation expressed as a single Handel-C statement will always execute in exactly one clock cycle, the execution time performance of the implementation can be improved by reducing the logic depth generated by the statement that generates the greatest logic depth. The logic depth generated by a statement can be reduced by decomposing complex logical operations into several less complex operations by introducing local registers to store intermediate results and consequently spreading the operation over multiple clock cycles. For example, in `FindBoundaryValueServer`, the process of comparing the list of ratios between the number of objects available and the number of objects required by a reaction rule for each object type is performed in a tree-like fashion with each level of the tree processed in one clock cycle.

The logic depth of a statement can also be reduced by avoiding the use of operations that generate very large combinatorial circuits. For example, to calculate the ratios in the `FindBoundaryValueServer`, one could use the standard Handel-C division operator. When implemented at the hardware level, the operation denoted by this operator

consumes only one clock cycle, but generates deep logic and uses significant hardware resources. As a consequence, alternative means of performing the division operation may be considered. One option is to implement the division operation using bit shifts. Because bit shifts can be implemented very efficiently on FPGAs, for our implementation the default setting is that division operations are implemented using bit shifts.

The algorithm in Figure 6.12 describes how the division operation is accomplished in our implementation using bit shifts. In Handel-C, each step in the algorithm can be implemented in one clock cycle without generating deep logic. So the total number of clock cycles taken to execute a division for two 8-bit variables is fixed at 25. Thus the advantage of this method is that a division can be performed in a fixed number of steps, each of which can be implemented in Handel-C in a fixed number of clock cycles.

As we have observed, reducing a path delay associated with the execution of an operation usually results in increasing the number of clock cycles taken to execute the operation. Thus, in the optimisation of a circuit, a trade-off needs to be made between minimising the number of clock cycles and maximising the clock rate. Hence careful judgement needs to be applied in attempting to satisfy the overall performance requirements for the circuit.

7 Evaluation of the hardware implementation of the DND algorithm

In this section, we evaluate our prototype hardware implementation of the DND algorithm. More specifically, we report experimental results related to the hardware resource consumption and clock rate of the implementation. Collectively, these results provide insight into the efficiency of the implementation, and suggest whether it is feasible to in-

Integer division algorithm

1. Initialise all the registers. The remainder has a width double the default width. Initialise the right
   half of the remainder register with the value of the dividend and initialise the left half of the remainder
   register with zeros.
2. Subtract the value of the divisor register from the left half of the remainder register.
3. If the value of the remainder is negative, restore the value of the remainder to its value prior to the
   subtraction operation occurring, perform a right-shift operation on the remainder, and then insert
   0 at the right end of the remainder register. Otherwise, perform a right-shift operation on the
   remainder, and insert 1 at the right end of the remainder register.

After n steps, where n is the default width, the remainder register contains the quotient in its right half and the remainder in its left half.

Fig. 6.12 A division algorithm using bit shifts.

Appli--cation	Regions	Rules	Object types × regions	Appli--cation	Regions	Rules	Object types × regions	Appli--cation	Regions	Rules	Object types × regions
A1	2	8	8	B1	4	16	8	C1	4	8	16
A2	4	16	16	B2	4	16	16	C2	4	16	16
A3	8	32	32	B3	4	16	32	C3	4	32	16
A4	16	64	64	B4	4	16	64	C4	4	64	16
A5	24	96	96	B5	4	16	96	C5	4	96	16

Table 7.2 Details of the input applications used in the experiments.

corporate the implementation into a hardware implementation of membrane computing such as Reconfig-P.

Before presenting the experimental results, we describe the experiments that have been conducted.

7.17 Details of the experiments Each experiment essentially involved the generation of hardware circuits for a particular instance of the object distribution problem, and the measurement of certain characteristics of these hardware circuits.

To perform the experiments, we developed two software programs. One program is able to automatically generate a set of linear systems (corresponding to a set of regions in a P system), where the coefficients and RHS constants of each linear system are set randomly. We refer to such a set of linear systems as an *input application*. The other program is able to analyse an input application and generate a Handel-C program that implements the DND algorithm for each linear system in the application.

Hardware circuits were generated for three classes of input applications. The applications in a given class are ordered according to their size as measured by a particular parameter. In the first class, applications differ with respect to the number of object types per region. In the second class, applications differ with respect to the number of reaction rules per region. And in the third class, applications differ with respect to the number of regions. The details of the input applications are given in Table 2.

Three circuits, and therefore three Handel-C programs, were generated for each input application. One of the circuits used block RAMs, whereas the other two circuits used distributed RAMs (the default option for memory). One of the two circuits using distributed RAMs used the standard Handel-C division operator, whereas the other circuit using distributed RAMs used the division algorithm specified in Figure 6.12. After compilation, each Handel-C program was synthesised into a circuit using Xilinx tools.

7.18 Experimental results Table 7.3 shows the results of the experiments. In the table, the values of various parameters are recorded for 45 hardware circuits (generated for the 15 input applications). The first of these parameters, the percentage of the

available LUTs used, serves as a measure of the overall hardware consumption of the implementation. The second of these parameters, the percentage of the available LUTs used as RAMs, serves as a measure of the extent to which hardware resources on the chip are used for data storage. In cases where block RAMs are used, the percentage of the available block RAMs used is recorded. For each circuit, the clock rate at which the circuit executes the input application is also recorded.

Influence of the type of scaling applied In terms of influence on hardware resource consumption, varying the number of regions has a greater effect than varying the number of object types, which in turn has a greater effect than varying the number of reaction rules. This is in accordance with our expectations. Since, to preserve system-level parallelism, an instance of the DND algorithm has to be implemented for every region, if the number of regions is increased, the hardware components that implement the DND algorithm need to be replicated once for each additional region. Consequently, the degree of hardware consumption increases approximately linearly with the number of regions. An increase in the number of object types results in a sublinear increase in hardware resource consumption mainly because of the fact that the hardware components that implement the major operations of the algorithm need to be replicated to allow parallel processing across object types. Since reaction rules (i.e., columns in the coefficient and traceback matrices) are processed one after the other, the relevant hardware components do not need to be replicated for the sake of parallel processing, and therefore increasing the number of reaction rules has a minimal effect on hardware resource consumption.

Influence of the type of RAMs used As expected, in general, circuits using distributed RAMs achieve higher clock rates than circuits using block RAMs. In terms of influence on hardware resource consumption, the percentage of the available LUTs used when distributed RAMs are used is approximately the same as the percentage used when block RAMs are used. However, it is noticeable that, when the number of reaction rules increases, the amount of block RAMs used remains constant. This is because each block RAM can store a large amount of data. In summary, it is generally advantageous to use distributed RAMs rather than block RAMs, but the use of block RAMs should perhaps be considered when there is a very large number of reaction rules.

Influence of the optimisation of the division operation In order to determine the effect of our optimisation of the division operation (see Section 6.16), for each application, both a circuit using the standard Handel-C division operation and a circuit using the bitshift division algorithm shown in Figure 6.12 were generated and compared. As illustrated in Table 7.3, circuits using the standard Handel-C division operation consume significantly more hardware resources (almost three times more in the case of application B5) and achieve lower clock rates than circuits using the bitshift division algorithm. This is largely due to the large combinatorial circuit that is generated by the standard Handel-C division operation. In summary, the results demonstrate that our introduction of an alternative to the Handel-C division operation is well motivated.

Overall assessment of the hardware resource consumption results The hardware resource consumption results are illustrated in graphical form in Figure 7.13.

As expected, the implementation of the DND algorithm consumes more hardware resources than the algorithm for deterministic object distribution used in Reconfig-P. Even so, in regard to our goal of incorporating nondeterministic object distribution into Reconfig-P, the hardware resource consumption results are promising. For example, executing the application with the largest number of regions (i.e., application A5) using the bitshift division algorithm requires only approximately 29% of the hardware resources to be used.

Overall assessment of the clock rate results In the default configuration (i.e., when distributed RAMs and bitshift division are used), the clock rates observed in the experiments range from 53 MHz to 77 MHz, with the average clock rate being 67 MHz. Incidentally, this average clock rate is almost exactly equal to the rate of the PCI bus which connects the host computer and FPGA used in Reconfig-P. Reconfig-P, in its current configuration, cannot exceed a clock rate of 66 MHz, regardless of the clock rate of the circuit being executed, because its clock rate is limited by the rate of the PCI bus. So, given that our ultimate goal is to incorporate the DND algorithm into Reconfig-P, the clock rate results are definitely promising.

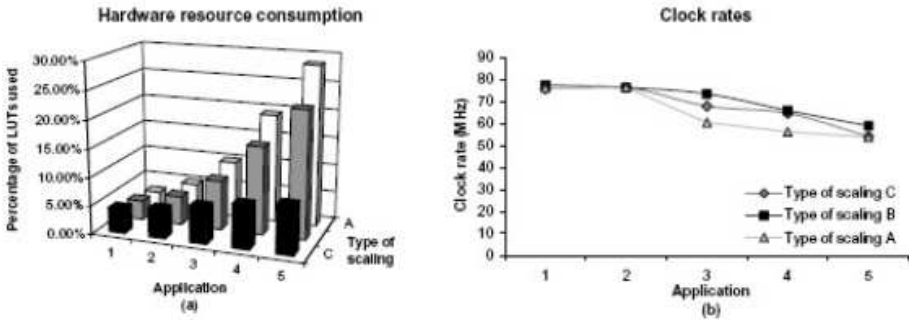


Fig. 7.13 Hardware resource consumption and clock rate results for the prototype hardware implementation of the DND algorithm in its default configuration.

8 Conclusion

We have devised an elegant and efficient algorithm for nondeterministic object distribution in P systems: the DND algorithm. We have also successfully implemented this algorithm in hardware using reconfigurable computing technology, thereby showing that it is suitable for implementation using such technology.

Application	Using distributed RAMs and bit-shift division (default configuration)			Using distributed RAMs and Handel-C division			Using block RAMs and bitshift division		
	% of LUTs	% LUTs as RAMs	Clock rate	% of LUTs	% LUTs as RAMs	Clock rate	% of LUTs	% of BRAMs	Clock rate
	Scaling number of regions								
A1	2.81%	0.10%	76	5.64%	0.10%	50	2.81%	5.55%	76
A2	5.10%	0.20%	76	10.77%	0.20%	48	5.28%	11.11%	71
A3	10.00%	0.40%	60	20.86%	0.40%	51	10.00%	22.22%	61
A4	19.32%	1.80%	56	40.96%	0.80%	46	19.50%	44.44%	57
A5	28.55%	1.21%	53	61.24%	1.21%	42	29.13%	66.66%	52
	Scaling number of object types								
B1	3.45%	0.11%	77	6.15%	0.11%	64	3.32%	5.55%	68
B2	5.10%	0.20%	76	10.77%	0.20%	48	5.28%	11.11%	71
B3	8.77%	0.40%	73	20.41%	0.39%	50	9.01%	22.22%	61
B4	15.57%	0.77%	65	38.54%	0.77%	46	15.16%	44.45%	60
B5	22.52%	1.15%	58	64.02%	1.15%	30	21.71%	66.66%	51
	Scaling number of reaction rules								
C1	4.37%	0.20%	75	6.70%	0.19%	53	4.29%	11.11%	68
C2	5.10%	0.20%	76	10.77%	0.20%	48	5.28%	11.11%	71
C3	6.49%	0.21%	67	12.50%	0.21%	48	6.39%	11.11%	66
C4	7.71%	0.21%	64	13.51%	0.21%	42	7.42%	11.11%	59
C5	8.54%	0.44%	53	14.00%	0.43%	41	8.06%	11.11%	52

Table 7.3 Hardware resource consumption and clock rate results for the prototype hardware implementation of the DND algorithm.

Our prototype hardware implementation of the DND algorithm realises the DND algorithm as a standalone process. Therefore we have not yet demonstrated that nondeterministic object distribution can be incorporated into Reconfig-P, which includes the other elements of basic cell-like P systems (such as updating of object multiplicities as a result of the application of reaction rules, and synchronisation of the applications of reaction rules occurring in different regions), without compromising too significantly its performance, flexibility or scalability. For example, it is conceivable that combining the existing standalone implementation of the DND algorithm with the existing

implementation of Reconfig-P will give rise to placement and routing problems that will significantly reduce the current clock rate and/or significantly increase the current hardware resource consumption of Reconfig-P. Nevertheless, our results provide strong evidence that nondeterministic object distribution can be incorporated into Reconfig-P as desired.

Our future work will involve an attempt to successfully incorporate the DND algorithm into Reconfig-P.

Bibliography

[1] Ciobanu, G. and Guo, W. 2004. P Systems Running on a Cluster of Computers. In Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G. and Salomaa, A. (eds) *Membrane Computing. International Workshop, WMC2003, Tarragona, Spain, July 2003. Revised Papers.* Vol. 2933 of Lecture Notes in Computer Science, Springer-Verlag, 123–139.

[2] George, M. and Alfke, P. 2007. Linear Feedback Shift Registers in Virtex Devices. `http://www.xilinx.com/bvdocs/appnotes/xapp210.pdf`, 269–285.

[3] Martinez, V., Fernandez, L., Arroyo, F. and Gutierrez, A. 2007. HW Implementation of a Optimized Algorithm for the Application of Active Rules in a Transition P-system. *International Journal Information Theories and Applications*, Vol. 14, 324–331.

[4] Nguyen, V., Kearney, D. and Gioiosa, G. 2007. Balancing Performance, Flexibility and Scalability in a Parallel Computing Platform for Membrane Computing Applications. In G. Eleftherakis et al. (eds) *WMC8 2007*. Vol. 4860 of Lecture Notes in Computer Science, Springer, pp. 385–413.

[5] Nguyen, V., Kearney, D. and Gioiosa, G. An Implementation of Membrane Computing using Reconfigurable Hardware. To appear in *Computing and Informatics*.

[6] Păun, G. 2002. *Membrane Computing: An Introduction*. Springer.

[7] Petreska, B. and Teuscher, C. 2004. A Reconfigurable Hardware Membrane System. In Martín-Vide, C., Mauri, G., Pãun, G., Rozenberg, G. and Salomaa, A. (eds) *Membrane Computing. International Workshop, WMC 2003, Tarragona, Spain, July 2003. Revised Papers*. Vol. 2933 of Lecture Notes in Computer Science, Springer-Verlag, 269–265.

Membrane algorithm solving job-shop scheduling problems

Taishin Y. Nishida, Tatsuya Shiotani, Yoshiyuki Takahashi

Toyama Prefectural University, Faculty of Engineering,
Imizu-shi, Toyama 939-0398, Japan
nishida@pu-toyama.ac.jp

In this paper, membrane algorithms solving job-shop scheduling problems (JSSP) are investigated. Two types of membrane algorithms are constructed by adding constraints of JSSP to components of membrane algorithms for solving travelling salesman problems. Computer simulations show that the algorithms can get considerably good approximations for several benchmark problems.

1 Introduction

Membrane algorithm [5–7] is a practical application of P system [9]. A membrane algorithm solves a combinatorial optimization problem using a framework from P system, i.e., a nested membrane structure. Each region which is separated by two consecutive membranes has a subalgorithm and a few tentative solutions of the problem. The subalgorithm converts the solutions into new solutions and then they are exchanged with adjacent regions. A better solution is sent to the inner region and a worse solution is sent to the outer region. Thus a well approximate solution will appear in the innermost region and it will be the output of the algorithm.

Membrane algorithm has two features, which other approximate algorithms seldom have. There is a spatial variety of subalgorithms in addition to (possibly) temporal variety. Many approximate algorithms, e.g., simulated annealing [2], change their parameters as the computation proceeds. On the other hand, membrane algorithm can assign various parameters to subalgorithms and let them run simultaneously. Secondly, membrane algorithm can combine subalgorithms based on different principles, e.g., genetic algorithm and simulated annealing. Using these features, a membrane algorithm has given good approximate solutions for the travelling salesman problem (TSP for short) [8].

In this paper, we apply membrane algorithm to the job-shop scheduling problems (JSSP for short). JSSP is a well known NP-hard optimization problem which is investigated

for about a half century [3]. Two types of membrane algorithms solving JSSP are constructed by modifying the algorithm for TSP [8], one has local search subalgorithms with temperature parameter (simulated annealing type) and a genetic algorithm with recombination only (no mutations) and the other has local search only. Computer simulations show that the algorithms can get considerably good approximations for several benchmark problems.

2 Job-shop scheduling problems

A job-shop scheduling problem (JSSP) consists of a number of jobs and a number of machines. In what follows we consider a JSSP with m machines $(1, \ldots, m)$ and n jobs $(1, \ldots, n)$. Each job i must be processed by all machines and each machine can process only one job at a time. A machine cannot be interrupted when it begins to process a job. Machine j finishes job i in a process time p_{ij}. For every job, an order of machines on which the job is processed is given by an instance of JSSP. The order is called a *technical order*. For every pair of job i and machine j, the process time p_{ij} is also given.

A schedule is assignments of beginning and ending times to all pairs of jobs and machines. The time to complete all jobs is the interval between the earliest beginning time and the latest ending time, which is called a *makespan*. Given technical order and process times p_{ij}, JSSP is to find a schedule of shortest makespan.

Table 2.1 A technical order (left) and process times (right) of 3×3 JSSP. Each entry of the left table shows the machine on which the job is to be processed at the order. The right table shows the process times p_{ij}, $1 \leq i \leq 3$ and $1 \leq j \leq 3$.

	order		
jobs	1st	2nd	3rd
1	1	2	3
2	1	3	2
3	2	3	1

	machines		
jobs	1	2	3
1	3	3	2
2	2	3	5
3	1	2	6

Example 1. Table 2.1 shows a technical order and process times of JSSP with 3 jobs and 3 machines.

A schedule is visualized by a Gantt chart. The left chart of Figure 2.1 shows a schedule in which each machine processes jobs in the order 1,2,3. The makespan of the left chart is 27. The right chart shows a schedule which gives the shortest makespan 15. There are more than one schedules which give the shortest makespan. □

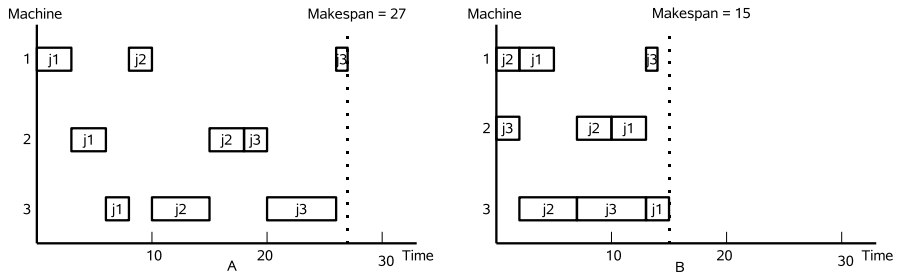


Fig. 2.1 Gantt charts of a schedule (A) and one of the optimal schedule (B).

Input: The solution matrix $S = (s_{ij})$.
Output: Solution matrix S' which is modified so that no deadlock occurs.
Symbols: processed[i] is a set of jobs which are processable on machine i. The initial value of processed[i] is $\{s_{i1}\}$ if job s_{i1} is processed on machine i at first by the technical order, $\emptyset$ otherwise.

1. Let $j = s_{ik}$ be the job which is examined whether it is processable or not.
   - 1.1 Let l be the order that job j is processed on machine i at l-th order in the technical order.
   - 1.2 If $l = 1$ (the top the the technical order) or $j \in \mathrm{processed}[t_{jl-1}]$ (where t_{jl-1} is the machine on which job j is processed at $l-1$-st order),
   - 1.3 then insert j into processed[i] (job j is processable on machine i).
2. If there are no jobs which become processable by steps 1.1—1.3 (there is a deadlock)
   - 2.1 Select i randomly from $1, \ldots, m$.
   - 2.2 Let $j = s_{ik}$ be the first job which is not processable on machine i.
   - 2.3 Let l be the order that job j is processed on machine i at l-th order in the technical order.
   - 2.4 $\forall x \in \{1, \ldots, l\}$ if $j \notin \mathrm{processed}[t_{jx}]$ (job j is not processable on machine t_{jx}) then do
     - 2.4.1 Let y be an integer such that $s_{t_{jx},y} = j$.
     - 2.4.2 Swap s_{ik} and $s_{t_{jx},y}$.
     - 2.4.3 Insert j into processed[t_{jx}].

Fig. 3.2 An outline of deadlock removing algorithm.

3 Design of membrane algorithm

We use an $n \times m$ matrix S to denote a solution of JSSP with n jobs and m machines. The i, j element s_{ij} of S denotes that job s_{ij} is processed on machine i at the j-th process of the machine. The beginning and ending times and the makespan are automatically

computed from S and the process times. For example, the next matrices S_1 and S_2 represent schedules A and B in Example 1

$$S_1 = \begin{pmatrix} 1\ 2\ 3 \\ 1\ 2\ 3 \\ 1\ 2\ 3 \end{pmatrix} \quad \text{and} \quad S_2 = \begin{pmatrix} 2\ 1\ 3 \\ 3\ 2\ 1 \\ 2\ 3\ 1 \end{pmatrix},$$

respectively.

Using the above notation, subalgorithms for a membrane algorithm solving JSSP are easily made from algorithms commonly used for other optimization problems, since solutions of many optimization problems are expressed in a vector or a matrix. In this paper, we make two subalgorithms which use one of the following principles.

The first one is a local search based on a random exchange. Let $v = (a_1, \ldots, v_k)$ be a vector. Then a vector $v' = (v_1, \ldots, v_{i-1}, v_j, v_{i+1}, \ldots, v_{j-1}, v_i, v_{j+1}, \ldots, v_k)$ where $1 \leq i < j \leq k$ are randomly selected integers is a neighbour of v.

The second one is an order crossover on two vectors which has been developed for genetic algorithm [4]. Since an order crossover is complicated, we explain it by an example. Let

$$A = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$$
$$B = (8, 7, 1, 2, 3, 10, 9, 5, 4, 6)$$

be vectors to be crossed over. Crossing positions, in this example 5 and 7, are randomly selected

$$A = (1, 2, 3, 4, \mid 5, 6, 7, \mid 8, 9, 10)$$
$$B = (8, 7, 1, 2, \mid 3, 10, 9, \mid 5, 4, 6).$$

We would like to insert the subsequence $(5, 6, 7)$ between the crossing positions of A to the corresponding part of B. In order to remove duplications (5,6,and 7) in the resulting vector $(8, 7, 1, 2, 5, 6, 7, 5, 4, 6)$, numbers 5, 6, and 7 in B are marked by H (and duplicated numbers in A are done so)

$$A = (1, 2, \mathrm{H}, 4, \mid 5, 6, 7, \mid 8, \mathrm{H}, \mathrm{H})$$
$$B = (8, \mathrm{H}, 1, 2, \mid 3, 10, 9, \mid \mathrm{H}, 4, \mathrm{H}).$$

The marked positions are filled with a sliding motion that starts following the second crossing position

$$A = (4, 5, 6, 7, \mid \mathrm{H}, \mathrm{H}, \mathrm{H}, \mid 8, 1, 2)$$
$$B = (2, 3, 10, 9, \mid \mathrm{H}, \mathrm{H}, \mathrm{H}, \mid 4, 8, 1).$$

Finally, subsequences between the crossing positions are placed and the new vectors are obtained

$$A' = (4, 5, 6, 7, |\, 3, 10, 9, |\, 8, 1, 2)$$
$$B' = (2, 3, 10, 9, |\, 5, 6, 7, |\, 4, 8, 1).$$

Using the above principles, we have designed Brownian and GA subalgorithms. A Brownian subalgorithm has one schedule matrix S. It makes a new matrix S' by the local search on one randomly selected row of S. If the makespan τ' of S' is smaller than the makespam τ of S, then S' is accepted as the solution that is sent to the adjacent regions. Otherwise, S' is accepted in probability $\exp(\tau - \tau')/T$ where T is a "temperature" parameter. If S' is not accepted, S becomes the solution to be sent. At the solution exchange phase, if the solution in the outer region is better than the solution in the inner region, then the two solutions are exchanged. In order for a solution not to go through all regions at one iteration of subalgorithms, exchanges between two Brownian regions are done every other iteration step.

A GA subalgorithm has two schedule matrices. The algorithm crosses every pair of corresponding rows of the matrices with the order crossover and makes two new matrices. The best solution of the four (old and new) solutions is sent to the inner region and the worst solution is sent to the outer region.

We must careful with deadlocks which are hidden in the matrix notation. In Example 1, the next matrix S_d cannot be processed

$$S_d = \begin{pmatrix} 3\ 1\ 2 \\ 2\ 3\ 1 \\ 1\ 2\ 3 \end{pmatrix}.$$

We denote $(M_x, J_x) <_T (M_y, J_y)$ (resp. $(M_x, J_x) <_S (M_y, J_y)$) if the combination of machine M_x and job J_x must be processed earlier than the combination of M_y and J_y by the technical order (resp. solution S). Then we have the next relations:

$$(M_1, J_3) <_{S_d} (M_1, J_2), \quad (M_3, J_2) <_{S_d} (M_3, J_3)$$
$$(M_1, J_2) <_T (M_3, J_2), \quad (M_3, J_3) <_T (M_1, J_3),$$

which make a loop $(M_1, J_3) < (M_1, J_3)$ or a deadlock. Although algorithms specific to JSSP use other data structures, e.g., a graph, we use matrix and a deadlock removing algorithm. Outline of the deadlock removing algorithm can be found in Figure 3.2.

4 Computer Experiments and their results

We examine two types of membrane algorithms. One has 20 membranes and a Brownian subalgorithm in each region. The temperature of the innermost region is 0. The temperature of region r ($0 < r < 20$) is given by $T_{max}\theta^{19-r}$ where

$$\theta = \sqrt[20]{\frac{T_{min}}{T_{max}}},$$

$T_{max} = 0.5mt_{max}$, and $T_{min} = 0.5t_{min}/m$ in which m is the number of machines and t_{max} and t_{min} are the maximum and minimum process time, respectively. This type is called "all B".

The other type has 20 membranes and one subalgorithm of genetic type and 19 Brownian subalgorithms. The innermost region (region 0) has Brownian with temperature 0, region 1 has genetic subalgorithm, and regions 2 to 19 have Brownian with temperature varying from 0 to the maximum. The temperature T of region r ($2 \leq r < 20$) is given by

$$T = \begin{cases} 0 & \text{if } r = 2 \\ T_{max}\theta^{19-r} & \text{otherwise} \end{cases}$$

where

$$\theta = \sqrt[18]{\frac{T_{min}}{T_{max}}}$$

and T_{max} and T_{min} are the same as the "all B" algorithm. The genetic subalgorithm in region 1 does the order crossover once in every 16 steps. It does nothing in every other 15 steps. Since a crossover changes solutions largely, crossover in every step is ill-balanced with local search. This algorithm is called "BG".

Table 4.2 shows the results for rather simple instances. Table 4.3 shows the results for more complex instances which are called the "10 tough problems" [1]. Every experiment consists of 20 trials. One trial is terminated after 200,000 iterations of subalgorithms at every region.

Table 4.2 Averages and standard deviations (in the parentheses) of results of membrane algorithms solving easy problems. The size is expressed by (number of jobs)×(number of machines). The number of trials is 20 for each algorithm-problem combination. MA (all B) stands for membrane algorithm with Brownian subalgorithms only and MA (BG) stands for membrane algorithm with Brownian and genetic subalgorithms. The "*" means that the optimum value is obtained.

Problems	(optimum, size)	MA (all B)	MA (BG)
abz5	(1235, 10×10)	1248.45 (3.23)	1247.55 (5.00)
la1	(666, 10×5)	666 (0.0)*	666 (0.0)*
la2	(655, 10×10)	655.15 (0.48)*	656.5 (3.04)*
la19	(842, 10×10)	850.2 (4.80)*	850.2 (4.12)*

Table 4.3 Best values, Averages, and standard deviations (in the parentheses) obtained by algorithms solving "10 tough problems". CBSA+SB stands for results by T. Yamada and R. Nakano [10]. Some optimum values are the best approximate values obtained up to now. The meaning of other symbols are identical to Table 4.2.

Problems	MA (all B)		MA (BG)		CBSA+SB	
(optimum, size)	best	av.(std.)	best	av.(std.)	best	av.(std.)
abz7 (656, 20×15)	706	715.4 (4.76)	701	716 (7.67)	665	671 (3.92)
abz8 (645, 20×15)	718	730.6 (5.92)	716	730.9 (7.73)	675	680 (3.13)
abz9 (661, 20×15)	727	745.6 (8.51)	728	743.0 (9.46)	686	698.6 (7.42)
la21 (1046, 15×10)	1059	1072.9 (7.71)	1058	1074.6 (8.83)	1046	1049.3 (3.32)
la24 (935, 15×10)	951	962.1 (5.03)	946	962.3 (7.89)	935	939.2 (1.99)
la25 (977, 15×10)	995	1007.5 (7.90)	988	1008.4 (9.72)	977	979.3 (1.62)
la27 (1235, 20×10)	1268	1279.3 (8.35)	1267	1276.0 (5.38)	1235	1242.4 (6.15)
la29 (1152, 20×10)	1202	1220.9 (11.3)	1196	1223.4 (13.4)	1154	1162.4 (7.10)
la38 (1196, 15×15)	1237	1265.0 (14.5)	1221	1251.7 (14.5)	1198	1206.8 (4.53)
la40 (1222, 15×15)	1262	1270.0 (6.09)	1249	1270.5 (12.3)	1228	1230.2 (2.31)

5 Conclusions

The membrane algorithms which are investigated in this paper cannot solve "tough problems" as well as a specially tuned algorithm [10]. The results do not discourage membrane algorithm because the membrane algorithms use common structures for optimization problems and only obey the JSSP specific constraint, i.e., deadlock. There must be many possibilities for improvement.

The combination of genetic and Brownian subalgirthms slightly better than Brownian only. This supports the advantage of membrane algorithm that it can combine subalgorithms based on different principles. Subalgorithms on various principles can widely search the solution space and hence can avoid a local minima.

Bibliography

[1] D. Applegate and W. Cook: A computational study of the job-shop scheduling problem, *ORSA J. on Comput.*, 3 (1991) 14-156.
[2] R. Azencott (eds): *Simulated Annealing*, (John Wily & Sons, New York, 1992).
[3] B. Giffler and G. L. Thompson: Algorithms for solving production-scheduling problems, *Operations Research*, 8 (1960) 487–503.
[4] D. E. Goldberg: *Genetic algorithms* (Addison-Wesley, Reading, 1989).

[5] T.Y. Nishida: An application of P-system: A new algorithm for NP-complete optimization problems, in *Proceedings of The 8th World Multi-Conference on Systems, Cybernetics and Informatics*, eds. N. Callaos et. al., (2004, vol V), pp. 109–112.

[6] T.Y. Nishida: An approximate algorithm for NP-complete optimization problems exploiting P-systems, in *Application of Membrane Computing*, eds. G. Ciobanu, Gh. Păun, and M.J. Pérez-Jiménez, (Springer-Verlag, Berlin, 2005), pp. 301–312.

[7] T.Y. Nishida: Membrane algorithm, in *Membrane Computing*, eds. R. Freund, Gh. Păun, G. Rozenberg, and A. Salomaa (Springer-Verlag, Berlin, 2006) pp. 55–66.

[8] T.Y. Nishida: Membrane algorithm with Brownian subalgorithm and genetic subalgorithm, *Foundations of Computer Science*, 18 (2007) 1353–1360.

[9] Gh. Păun: Computing with membranes, *Journal of Computer and System Sciences*, 61 (2000), 108–143.

[10] T. Yamada and R. Nakano: Job-shop scheduling by simulated annealing combined with determinisitc local search, *Journal of IPSJ*, 37 (1996) 597–604 (in Japanese).

On mathematical modeling of anatomical assembly, spatial features, and functional organization of cortex by application of hereditarily finite sets

Adam Obtulowicz

Polish Academy of Sciences, Institute of Mathematics,
Śniadeckich 8, P.O.B. 21, 00-956 Warsaw, Poland
A.Obtulowicz@impan.gov.pl

A common ground language for linking anatomical, spatial and functional organization aspects of cortex is proposed with regards to conceptual (descriptive) and computational approaches to cortex models formation, including distributed computational systems inspired by cortex models. The core of that common ground language is mathematical language of hereditarily finite sets, and some known cortex models are described in this language of hereditarily finite sets to link the mentioned cortex aspects.

1 Introduction

We propose a common ground language for linking anatomical, spatial, and functional organization aspects of cortex, where the regards to conceptual (descriptive) and computational approaches to cortex models formation are respected.

The regard to conceptual approach, contained in the paper, concerns anatomical assembly, spatial features, and functional organization of cortex which are based on cortex models proposed by V. B. Mountcastle in [27], [28] and by D. C. Van Essen in [15], [39]. In the last two papers functional organization of cortex has a form of hierarchical system then developed and improved in [20], [21].

The regard to computational approach, contained in the paper, concerns computational models of cortex based on the concept of a system of nested neural networks described by J. P. Sutton et al. in [37] and on the idea of a network of networks due to J. A. Anderson and J. P. Sutton, presented in [26], [5], [6], where distributed computational systems inspired by cortex models are also proposed.

The core of the proposed common ground language is mathematical language of hereditarily finite sets which was applied by R. Gandy in [17] to give a characterization of

abstract computing devices (the characterization was then improved and more extensively explained by W. Sieg in [34], [35]) and by N. De Pisapia in [11] to model abstract neural nets.

The cortex models presented in the papers quoted above are described in the language of hereditarily finite sets in Sections 2 and 3 of the present paper, where the links between the mentioned cortex aspects are also discussed.

The paper is aimed among others to be a hint and a brief survey of some methods of cortex modeling and related topics for those who will approach problem B in [29] of applications of spiking neural P systems (introduced in [22]) in neurology, in particular in modeling cortex behaviour and computations inspired by this behaviour.

The basic concepts concerning hereditarily finite sets are introduced and explained in the following definitions and comments.

Definition 1 *We recall now the notion of a hereditarily finite set used in [17]. For a potentially infinite set L of labels or names which are urelements, i.e., they are not (treated as) sets themselves, we define inductively a family of sets HF_i for natural numbers $i \geq 0$ such that*

$$\begin{aligned} \mathrm{HF}_0 &= \varnothing, \\ \mathrm{HF}_{i+1} &= \textit{the set of nonempty finite subsets of } L \cup \mathrm{HF}_i. \end{aligned}$$

The elements of the union $\mathrm{HF} = \bigcup\{\mathrm{HF}_i \,|\, i \geq 0\} \cup \{\varnothing\}$ are called hereditarily finite sets over L or hereditarily finite sets with urelements in L, or simply hereditarily finite sets if there is no risk of confusion.

For $x \in \mathrm{HF}$ we define its weak transitive closure $\mathrm{WTC}(x)$ and support $\mathrm{supp}(x)$ by

$$\begin{aligned} \mathrm{WTC}(x) &= \bigcup\big\{\mathrm{WTC}(y) \,|\, y \in x \textit{ and } y \in \mathrm{HF}\big\} \cup \{x\} \\ \mathrm{supp}(x) &= (x \cap L) \cup \bigcup\{\mathrm{supp}(y) \,|\, y \in x \textit{ and } y \in \mathrm{HF}\}, \end{aligned}$$

and the depth of x is defined to be the smallest natural number i for which $x \in \mathrm{HF}_i$.

We write $\mathrm{depth}(x)$ to denote the depth of a hereditarily finite set x.

Explanatory Comments 1 *One interprets a hereditarily finite set x of depth greater than 1 and the corresponding sets $\mathrm{WTC}(x)$ and $\mathrm{supp}(x)$ in the following way.*

The urelements belonging to $\mathrm{supp}(x)$ are elementary or indecomposable pieces of x, the elements of $\mathrm{WTC}(x) - \{x\}$ are composed pieces of x. The set x itself is assembled, or composed, or aggregated successively from these two kinds of pieces elementary and complex one according to membership relation $\in$ restricted to

$\mathrm{WTC}(x) \cup \mathrm{supp}(x)$. *Any (pictorial or verbal) presentation of this restriction of* $\in$ *may serve as a plan or an algorithm of an assembly of* x*. If* x *is a model of a system, e.g., an organ in biology, the elements of* $\mathrm{supp}(x)$ *correspond to indecomposable pieces of this system and the elements of* $\mathrm{WTC}(x) - \{x\}$ *correspond to composed pieces of this system.*

Explanatory Comments 2 *For a hereditarily finite set* x *the collection* $\{\mathrm{supp}(y) \,|\, y \in \mathrm{WTC}(x)\}$ *can represent spatial aspect of* x *as a collection of sets of regions of a space (on topological level of abstraction) in which pieces of* x *are placed. Hence the assignment* $y \mapsto \mathrm{supp}(y)$ *(*$y \in \mathrm{WTC}(x)$*) links assembly aspect of* x *represented by* $\mathrm{WTC}(x)$ *with spatial aspect of* x *represented on topological level of abstraction by* $\{\mathrm{supp}(y) \,|\, y \in \mathrm{WTC}(x)\}$.

2 Columnar and areal organization of cortex

In this section we discuss anatomical, spatial, and functional organization aspects of cortex with a regard to conceptual approach to cortex models formation. We show how cortex can be modelled in terms of hereditarily finite sets to provide linking of these aspects.

We begin the discussion with anatomical aspect of cortex. The columnar and areal organization of cortex described in [27], [39], discussed also in [8], [10], [23], [24], and the proposed interpretation of hereditarily finite sets in Explanatory Comments 1 give rise to the following main example of hereditarily finite sets.

Main example 1 *Anatomical assembly of cortex with respect to its columnar and areal organization can be modelled by hereditarily finite set* x_{cortex} *of depth 4 such that*

- x_{cortex} *itself is the set of all areas of cortex,*
- *areas of cortex are non-empty sets of hypercolumns of cortex,*
- *hypercolumns are non-empty sets of columns of cortex,*
- *columns are non-empty sets of neurons of cortex, where neurons are treated as urelements.*

We will discuss hypothetical properties of x_{cortex} by using the following auxiliary notions.

Definition 2 *A collection of sets forms a tree if and only if, for any two sets that belong to the collection, either one is wholly contained in the other, or else they are wholly disjoint. See [3].*

Remark 1 *A lattice theoretical treatment of the above defined trees relates them in [25] to ultrametric spaces, where a relationship is described by an isomorphism of appropriate categories mentioned directly in the title of [25]. For a survey of applications of ultrametric spaces in biology and physics see e.g. [31].*

Definition 3 *For two elements x, y of a collection $\mathcal{A}$ of sets we say that x is an immediate subset of y with respect to $\mathcal{A}$ if x is a proper subset of y and there does not exist a set z in $\mathcal{A}$ such that x is a proper subset of z and z is a proper subset of y.*

Definition 4 *A hereditarily finite set x is a tree-like hereditarily finite set if the assignment $y \mapsto \mathrm{supp}(y)$ ($y \in \mathrm{WTC}(x)$) is an injection and $\{\mathrm{supp}(y) \,|\, y \in \mathrm{WTC}(x)\}$ is a tree.*

Definition 5 *We say that a hereditarily finite set x is a regular hereditarily finite set if the assignment $y \mapsto \mathrm{supp}(y)$ ($y \in \mathrm{WTC}(x)$) is an injection and $z \in y$ implies that $\mathrm{supp}(z)$ is an immediate subset of $\mathrm{supp}(y)$ with respect to $\{\mathrm{supp}(y) \,|\, y \in \mathrm{WTC}(x)\}$ for all $y, z \in \mathrm{WTC}(x)$.*

Definition 6 *We define depth-homogeneous hereditarily finite sets by induction in the following way:*

- *a depth-homogeneous hereditarily finite set of depth 1 is a finite non-empty set of urelements,*
- *a depth-homogeneous hereditarily finite set of depth $n+1$ is a finite non-empty set of depth-homogeneous hereditarily finite sets of depth n.*

Artificial example 1 *We present an example of depth-homogeneous hereditarily finite sets of depth 3:*

$x = \big\{\{\{1,5\},\{1,4\},\{1,3\}\},\{\{2,4\},\{1,3\}\},\{\{2,4\},\{2,5\}\}\big\}$ is a regular hereditarily finite set but the collection $\{\mathrm{supp}(y) \,|\, y \in \mathrm{WTC}(x)\}$ is not a tree and hence x is not a tree-like hereditarily finite set.

Theorem 2.1 *Every tree-like hereditarily finite set is a regular hereditarily finite set but the converse is not true, i.e., there exist regular hereditarily finite sets which are not tree-like hereditarily finite sets.*

Proof One proves the theorem by induction on depth of hereditarily finite sets. The set x in Artificial Example 1 is a regular hereditarily finite set which is not a tree-like hereditarily finite set. □

Comment 1 *A collection of sets which is a tree is a form of an idealized spatial (on topological level of abstraction) organization of a system mostly to simplify system analysis or system recurrent construction than to be a result of an evolutive natural process of spatial adaptation of a system. This observation due to Ch. Alexander is contained in his paper [3] concerning applications of tree structures in city planning. Thus since by Theorem 2.1 the notion of a regular hereditarily finite set is less restrictive than the notion of a tree-like hereditarily finite set, we propose the following hypothesis.*

Hypothesis 1 *The hereditarily finite set x_{cortex} modelling cortex is a depth-homogeneous, regular hereditarily finite set.*

The assignment $y \mapsto \mathrm{supp}(y)$ ($y \in \mathrm{WTC}(x)$) establishes the links between anatomical assembly aspect and spatial aspect of cortex for $x = x_{\mathrm{cortex}}$, see Explanatory Comments 2.

Functional organization aspect of cortex is modelled in [15], [39], [20], [21] by using more or less explicitly a graph of pathway connections between cortex areas. The vertices of that graph of pathway connections are areas (or their labels, or names), the edges are those ordered pairs of areas which are determined, among others, by functions $\mu_{y,y'} : \mathrm{supp}(y) \times \mathrm{supp}(y') \to \mathbb{R}$ ($(y, y') \in x_{\mathrm{cortex}} \times x_{\mathrm{cortex}}$) valued in the set $\mathbb{R}$ of real numbers, where the values $\mu_{y,y'}(n, n')$ describe strength of synaptic connection between neurons $n \in \mathrm{supp}(y)$ and $n' \in \mathrm{supp}(y')$. The graph of pathway connections determined by functions $\mu_{y,y'}$ ($(y, y') \in x_{\mathrm{cortex}} \times x_{\mathrm{cortex}}$) describing strength of synaptic connections links functional organization aspect of cortex with its anatomical and spatial aspects.

3 Multilevel (nested) neural networks and networks of networks

In this section we discuss certain topic computational models of cortex which are based on the concept of a system of nested neural networks described by J. P. Sutton et al. in [37] and on the idea of a network of networks due to J. A. Anderson and J. P. Sutton presented in [26], [5], [6].

We describe the concept of a system of nested neural networks and the idea of a network of networks in terms of hereditarily finite sets by using the following notion.

Definition 7 *For a natural number $n > 1$ an n-level network $\mathfrak{N}$ is given by*

- *a depth-homogeneous tree-like hereditarily finite set $x^{\mathfrak{N}}$ of depth n, called the underlying hereditarily finite set of $\mathfrak{N}$,*

- *a family of state interaction functions*[13] $\mu^y_{z,z'} : \mathrm{supp}(z) \times \mathrm{supp}(z') \to \mathbb{R}$ *(*$(z, z') \in (y \times y) - \dot{\Delta}_y$ *and* $y \in \mathrm{WTC}(x^{\mathfrak{N}})$ *with* $\mathrm{depth}(y) > 1$*) valued in the set* $\mathbb{R}$ *of real numbers, where* $\dot{\Delta}_y = \{(z, z) \,|\, z \in y\}$ *for* $\mathrm{depth}(y) > 2$ *and* $\dot{\Delta}_y = \varnothing$ *for* $\mathrm{depth}(y) = 2$*,*
- *two functions* $\sigma^{\mathfrak{N}}, \theta^{\mathfrak{N}} : \mathrm{supp}(x^{\mathfrak{N}}) \to \mathbb{R}$ *which are state function and threshold function of* $\mathfrak{N}$*, respectively.*

The underlying hereditarily finite set $x^{\mathfrak{N}}$ of an n-level network $\mathfrak{N}$ describes hierarchical organization of $\mathfrak{N}$, where the elements of $\mathrm{WTC}(x^{\mathfrak{N}})$ correspond to clusters, see [37], and indecomposable units of $\mathfrak{N}$ are urelements belonging to $\mathrm{supp}(x^{\mathfrak{N}})$, e.g., neural elements themselves or elementary processors. The state interaction functions $\mu^y_{z,z'}$ describe the strength of synaptic connections between neurons like functions $\mu_{y,y'}$ in Section 2.

Corollary 1 *Let* $\mathfrak{N}$ *be an* n*-level network. Then*

- *for* $n = 2$ *the network* $\mathfrak{N}$ *is a network of networks understood as in* [26], [5]*,*
- *for* $n > 2$ *and* $y \in x^{\mathfrak{N}}$ *one obtains an* $(n-1)$*-level network* $\mathfrak{N}[y]$ *determined by* y *such that* y *is the underlying hereditarily finite set of* $\mathfrak{N}[y]$*, the family of state interaction function of* $\mathfrak{N}[y]$ *is the restriction of the family of state interaction functions of* $\mathfrak{N}$ *to the case of* $\mathrm{WTC}(y)$*, and the state function with threshold function of* $\mathfrak{N}[y]$ *are the restrictions of the state function and the threshold function of* $\mathfrak{N}$*, respectively, to the set* $\mathrm{supp}(y)$*.*

Thus for $n > 2$ *an* n*-level network* $\mathfrak{N}$ *is a network of* $(n-1)$*-level networks* $\mathfrak{N}[y]$ $(y \in x^{\mathfrak{N}})$ *of* $(n-2)$*-level networks* $\mathfrak{N}[y][z]$ $(z \in y)$*, etc., where* $\mathfrak{N}[y]$ *is immediately nested in* $\mathfrak{N}$ *and* $\mathfrak{N}[y][z]$ *is immediately nested in* $\mathfrak{N}[y]$*.*

Proof The corollary is an immediate consequence of the definition of an n-level network. □

Theorem 3.1 *For a natural number* $n > 1$*, let* $\mathfrak{N}$ *be an* n*-level network. Then there exists a unique function* $\mu^* : \mathrm{supp}(x^{\mathfrak{N}}) \times \mathrm{supp}(x^{\mathfrak{N}}) \to \mathbb{R}$ *such that* $\mu^* \restriction \mathrm{supp}(z) \times \mathrm{supp}(z') = \mu^y_{z,z'}$ *for all* $(z, z') \in y \times y - \dot{\Delta}_y$*,* $y \in \mathrm{WTC}(x^{\mathfrak{N}})$*, where* $\mu^* \restriction \mathrm{supp}(z) \times \mathrm{supp}(z')$ *denotes the restriction of* μ^* *to the set* $\mathrm{supp}(z) \times \mathrm{supp}(z')$*.*

Proof We prove the theorem by induction on n. □

[13] corresponding to state interaction matrices in [5]

Corollary 2 *For a natural number $n > 1$ an n-level network $\mathfrak{N}$ behaves globally as a usual neural network whose state interaction function is that μ^* which is given in Theorem* 3.1.

Proof The corollary is an immediate consequence of Theorem 3.1. □

Remark 2 *Assume that we are given a neural network whose hierarchical organization is represented by a depth-homogeneous tree-like hereditarily finite set x, where $\mathrm{supp}(x)$ is the set of neurons of the network and $\{\mathrm{supp}(y) \,|\, y \in \mathrm{WTC}(x)\}$ is the collection of network regions. The network may contain neighbouring network regions $\mathrm{supp}(z)$, $\mathrm{supp}(z')$ with $\{z, z'\} \subset y \in \mathrm{WTC}(x)$ for some y, where for all neurons $n \in \mathrm{supp}(z)$ and $n' \in \mathrm{supp}(z')$ there exists a synaptic connection between n and n'. And vice versa, the network may contain regions $\mathrm{supp}(z)$, $\mathrm{supp}(z')$ which are far one to another such that for all neurons $n \in \mathrm{supp}(z)$ and $n' \in \mathrm{supp}(z')$ there does not exist any synaptic connection. The above two cases of synaptic connection and its lack give rise to the following definition.*

Definition 8 *By an n-level network with neighbourhood graphs we mean an n-level network $\mathfrak{N}$ except it is completed by a new data which are neighbourhood graphs G^y ($y \in \mathrm{WTC}(x)$ with $\mathrm{depth}(y) > 1$) and the family of state interaction functions is restricted by the neighbourhood graphs such that*

- *the set $V(G^y)$ of vertices of G^y is y itself, the set $E(G^y)$ of edges of G^y is such that $E(G^y) \subset (y \times y) - \dot{\Delta}_y$ for $\dot{\Delta}_y$ given as in Definition 7,*
- *the family of state interaction functions $\mu^y_{z,z'} : \mathrm{supp}(z) \times \mathrm{supp}(z') \to \mathbb{R}$ is determined by neighbourhood graphs such that $(z, z') \in E(G^y)$ and $y \in \mathrm{WTC}(x^{\mathfrak{N}})$ with $\mathrm{depth}(y) > 1$.*

The neighbourhood graphs are interpreted such that for $(z, z') \in (y \times y) - \dot{\Delta}_y$ but with $(z, z') \notin E(G^y)$ certainly for all neurons $n \in \mathrm{supp}(z)$ and $n' \in \mathrm{supp}(z')$ there does not exist any synaptic connection from n into n'.

For an n-level network with neighbourhood graphs its neighbourhood graphs can be illustrated by a Venn diagram of frontier disjoint balls, where the balls represent the elements of the weak transitive closure of the underlying hereditarily finite set of the network and the edges of the neighbourhood graphs are represented by the arcs connecting the frontiers of the corresponding balls, e.g. Figure 10 in the paper [5] illustrates the neighbourhood graph of corresponding 2-level network, see also Figure 1 in [37] illustrating neighbourhood graphs of a 3-level network.

Corollary 3 *Let $\mathfrak{N}$ be an n-level network with neighbourhood graphs G^y ($y \in \mathrm{WTC}(x^{\mathfrak{N}})$ with $\mathrm{depth}(y) > 1$). Then $\mathfrak{N}$ determines an inductive construction of a family of graphs*

G_y ($y \in \mathrm{WTC}(x^{\mathfrak{N}})$) with $G_{x^{\mathfrak{n}}}$ as a final result such that the set $V(G_y)$ of vertices of G_y and the set $E(G_y)$ of edges of G_y are given by

- $V(G_y) = \mathrm{supp}(y)$ *for all* $y \in \mathrm{WTC}(x^{\mathfrak{N}})$,
- *for* $y \in \mathrm{WTC}(x^{\mathfrak{N}})$ *with* $\mathrm{depth}(y) = 1$

$$E(G_y) = \big\{(a,b) \in V(G_y) \times V(G_y) \,|\, \mu^{y'}_{y,y}(a,b) > 0\big\} \quad \textit{if } (y,y) \in E(G^{y'}),$$

otherwise $E(G_y) = \varnothing$, where y' is that unique element of $\mathrm{WTC}(x^{\mathfrak{N}})$ *for which* $y \in y'$,

- *for* $y \in \mathrm{WTC}(x^{\mathfrak{N}})$ *with* $\mathrm{depth}(y) > 1$

$$E(G_y) = \bigcup \Big\{ \big\{(a,b) \in V(G_y) \times V(G_y) \,|\, \mu^{y}_{z,z'}(a,b) > 0\big\} \;\Big|\; (z,z') \in E(G^y) \text{ and } z \neq z' \Big\} \cup \bigcup \big\{E(G_z) \,|\, z \in y\big\}.$$

Proof The corollary is an immediate consequence of the definition of an n-level network with neighbourhood graphs. □

Remark 3 *The construction in Corollary 3 is a generalization of the constructions of an n-dimensional hypercube from the copies of hypercubes of dimensions less than n for $n > 3$. For some mathematical treatment of n-dimensional hypercubes see e.g. [12] and for their applications see e.g. [33]. Some of the constructions of higher dimensional hypercubes from the copies of hypercubes of low dimensions were pointed out by Richard Feynman and Tamiko Thiel to apply these constructions among others for visual presentation of 'constructive' structure of higher dimensional hypercubes in [38]. The drawings of structures of 6-D hypercube, 9-D hypercube, and 12-D hypercube in Chapter II of [38] can be treated intuitively as illustrations of those 2-level, 3-level, and 4-level networks with neighbourhood graphs, respectively, which determine the constructions (like in Corollary 3) of 6-D hypercube, 9-D hypercube and 12-D hypercube from the copies of 3-D cube, respectively. The neighbourhood graphs of those 2-level, 3-level and 4-level networks are isomorphic to 3-D cube.*

Remark 4 *This remark is a hint for those who will approach modeling cortex behaviour by application of spiking neural P systems. The large number (about 10^{11}) of neurons in human cortex and various structural cortex organizations (anatomical, physiological, and functional one) suggest to approach modeling of cortex behaviour by application of (higher level) networks of spiking neural P systems in a similar way as networks of networks are applied, cf. [6]. It is worth to investigate those spiking neural P systems whose sets of neurons and synapses form synfire braids and chains, cf. [7], [14]. The discussion illustrated by Figure 16 in Section 5 of [1] suggests that a concept of an n-level network of spiking neural P systems with their sets of neurons and*

synapses forming synfire chains may appear useful to model compositionality ability of brain by binding synfire chains. Namely, one may consider those higher level constructs (similar to construct illustrated by Figure 16 in Section 5 in [1]) which are described in terms of spiking neural P systems whose neurons and synapses form an n-level network with neighbourhood graphs, where the levels of its hierarchical organizationation (with respect to nesting relation in Corollary 1) may correspond to the abstraction levels of image perceiving and processing from pixel-level, then local feature level, structure (edge) level, object level, to object set level and scene characterization level. It gives rise to an open problem, which is a variant of Temporal Correlation Hypothesis of Visual Feature Integration formulated e.g. in [18], whether hierarchical organizationation of those higher level constructs is determined by spiking trains reached in those learning processes, e.g. in [19], which respect hierarchical organizationation like in [13].

Conclusion. The common ground language for linking various aspects of cortex models, proposed in the paper, concerns mainly the models respecting columnar and areal organization of cortex but neuroscience and related fields contain a lot of other models different from those which are based on columnar and areal cortex organization. Looking forwards, the proposed common ground language could be a node of a future net (or web) of common ground languages aimed to provide a discourse between methods, treatments, and approaches, all respecting various motivations, to cortex models formation from:

- cortex models due to E. Bienestock in [7] based on synfire chains and braids,
- the models in [32] extending the hierarchical organization of cortex pointed out by D. H. Hubel and T. N. Wiesel, where there are specified the level of simple cells (neurons), the level of complex cells, and the level of hypercomplex cells such that a complex cell responds to some assembly of simple cells, and a hypercomplex cell responds to some assembly of complex cells,
- neural net models using tensor product in [30], [36], and motivated by problems of linguistics,
- the models of brain memory inspired by the idea of spin glass models in physics, see [31] and Introduction to [37],

to the models inspiring construction of distributed systems realizing massively parallel computations, cf. [6], [26], and neoperceptron due to K. Fukushima [16].

That future net could be a platform for mutual inspiration between neuroscience and other disciplines and fields of research and engineering, e.g., city planning, where general aspects of harmony-seeking computations and evolutive processes of spatial adaptation are discussed, cf. [2].

Maybe one could form that future net of common ground languages in a similar way to the networks of patterns forming pattern languages, due to Ch. Alexander, discussed

in [4]. It is worth to point here that the idea of pattern languages has already inspired computer scientists to similar constructions in the area of object programming, cf. [9].

Bibliography

[1] Abeles, M., Hayon, G., Lehmann, D., *Modeling compositionality by dynamic binding synfire chains*, Journal of Computational Neuroscience 17 (2004), pp. 179–201.

[2] Alexander, Ch., *Harmony-Seeking Computations*, to be published in the International Journal of Unconventional Computation, see also `http://www.livingneighborhoods.org`

[3] Alexander, Ch., *A city is not a tree*, Architectural Forum 122 (1965), No. 1, pp. 58–61; No. 2, pp. 58–62; `http://www2.rudi.net/bookshelf/classics/city`

[4] Alexander, Ch., *Pattern Languages*, Oxford Univ. Press, New York 1977.

[5] Anderson, J. A., *Arithmetic on a Parallel Computer*: *Perception Versus Logic*, Brain and Mind 4 (2003), pp. 169–188.

[6] Anderson, J. A., *A Brain-Like Computer for Cognitive Software Applications*: *The Ersatz Brain Project*, `http://www.ErsatzBrain.org`

[7] Bienestock, E., *A model of neocortex*, Network: Computation in Neural Systems 6 (1995), pp. 179–224.

[8] Buxhoeven, D. P., Casanova, M. F., *The minicolumn hypothesis in neuroscience*, Brain 125 (2002), pp. 935–951.

[9] Coplien, J. O., *Software Patterns*, SIGS Books & Multimedia, New York 2000.

[10] Cürüklü, B., Lansner, A., *An Abstract Model of a Cortial Hypercolumn*, in: Proc. of International Conference on Neural Information Processing (ICONIP'02), Singapore, IEEE 2002, pp. 80–85.

[11] De Pisapia, N., *Gandy Machines*: *an Abstract Model for Parallel Computations, for Turing Machines, the Game of Life, and Artificial Neural Networks*, M.S. Thesis, Carnegie Mellon Univ., Pittsburgh 2000, `http://artsci.wustl.edu/~ndepisap`

[12] Domshlak, C., *On recursively directed hypercubes*, Electron. J. Combin. 9 (2002), #R23.

[13] Dotsenko, V. S., *Hierarchical model of memory*, Physica 140A (1986), pp. 410–415.

[14] Doursat, R., Bienestock, E., *Neocortial self-structuration as a basis for learning*, in: 5th International Conference on Development and Learning (ICDL 2006), Bloomington, Indiana (to appear).

[15] Felleman, D. J., Van Essen, D. C., *Distributed hierarchical processing in the primate cerebral cortex*, Cerebral Cortex 1 (1991), No. 1, pp. 1–47.

[16] Fukushima, K., *Neocognitron*: *A hierarchical neural network capable of visual pattern recognition*, Neural Networks 1 (1988), No. 2, pp. 119–130.

[17] Gandy, R., *Church's thesis and principles for mechanisms*, in: The Kleene Symposium, eds. J. Barwise et al., North-Holland, Amsterdam 1980, pp. 123–148.

[18] Gray, C. M., *The temporal correlation hypothesis of visual feature integration still alive and well*, Neuron 24 (1999), pp. 31–47.

[19] Gutierez-Naranjo, M. A., and Perez-Jimenez, M. J., *A spiking neural P systems based model for Hebbian learning*, this volume.

[20] Hilgetag, C.-C., O'Neill, M. A., Young, M. P., *Indeterminate organization of the virtual system*, Science 271 (1996), pp. 776–777.

[21] Hilgetag, C.-C., O'Neill, M. A., Young, M. P., *Hierarchical organization of macaque and cat cortial sensory systems explored with a novel network processor*, Phil. Trans. Royal Soc. London, B 355 (2000), pp. 71–89.

[22] Ionescu, M., Păun, Gh., Yokomori, Y., *Spiking neural P systems*, Fund. Inform. 71 (2006), pp. 279–308.

[23] Johansson, Ch., *An Attractor Memory Model of Neocortex*, Ph.D. Thesis, Royal Institute of Technology, Stockholm 2004.

[24] Johansson, Ch., Lansner, A., *Towards cortex sized artificial nervous systems*, in: Proc. Knowledge-Based Intelligent Information and Engineering Systems, KES'04, Lecture Notes in Artificial Intelligence 3213, Springer, Berlin 2004, pp. 959–966.

[25] Lemin, A. J., *The category of ultrametric spaces is isomorphic to the category of complete, atomic, tree-like, and real graduate lattices* LAT^*, Algebra Universalis 50 (2003), pp. 35–49.

[26] Ling Guan, Anderson, J. A., Sutton, J. P., *A network of networks processing model for image regularization*, IEEE Transactions on Neural Networks 8 (1997), No. 1, pp. 169–174.

[27] Mountcastle, V. B., *The columnar organization of the neocortex*, Brain 120 (1997), pp. 701–722.

[28] Mountcastle, V. B., Introduction to special issue on cortial columns, Cerebral Cortex 13 (2003), No. 1, pp. 2–4.

[29] Păun, Gh., Perez-Jimenez, M. J., *Spiking neural P systems. Recent results, research topics*, in: 6th Brainstorming Week on Membrane Computing, Sevilla 2008, web page.

[30] Prince, A., Smolensky, P., *Optimality*: *from neural networks to universal grammar*, Science 275 (1997), 14 March, pp. 1604–1610.

[31] Rammal, R., Toulouse, G., Virasoro, M. A., *Ultrametricity for physicists*, Rev. Modern Phys. 58 (1986), pp. 765–788.

[32] Reisenhuber, M., Poggio, T., *Hierarchical models of object recognition in cortex*, Nature Neuroscience 11 (1999), pp. 1019–1025.

[33] Seitz, Ch. L., *The cosmic cube*, Comm. ACM 28 (1985), pp. 22–33.

[34] Sieg, W., *Computability Theory*, Seminar Lectures, University of Bologna, November 2004,
`http://www.phil.cmu.edu/summerschool/2006/Sieg/computability_theory.pdf`

[35] Sieg, W., *Calculations by man and machine*: *conceptual analysis*, Lecture Notes in Logic 15, Berlin 2002, pp. 390–409.

[36] Smolensky, P., *Tensor product variable binding and the representation of symbolic structures in connectionist systems*, Artificial Intelligence 46 (1990), pp. 159–216.
[37] Sutton, J. P., Beis, J. S., Trainor, L. E. H., *Hierarchical model of memory and memory loss*, J. Phys. A: Math. Gen. 21 (1988), pp. 443–445.
[38] Thiel, T., *The design of the connection machine*, Design Issues 10 (1994), pp. 5–18; see also `http://www.mission-base.com/tamiko/cm/cm_articles.html`.
[39] Van Essen, D. C., Maunsell, J. H. R., *Hierarchical organization and functional streams in the virtual cortex*, Trends in NeuroScience, September 1983, pp. 370–375.

Toward a wet implementation for τ-DPP

Dario Pescini, Paolo Cazzaniga, Claudio Ferretti, Giancarlo Mauri

Università degli Studi di Milano-Bicocca,
Dipartimento di Informatica, Sistemistica e Comunicazione,
Viale Sarca 336, 20126 Milano, Italy
{cazzaniga, ferretti, mauri, pescini}@disco.unimib.it

In the last decade, different computing paradigms and devices inspired by biological and biochemical systems have been proposed. Here, we recall the notions of membrane systems and the variant of τ-DPP. We introduce the framework of chemical computing, in order to show how to describe computations by means of a chemical reaction system. Besides the usual encoding of primitive boolean functions, we also present encodings for instructions of register machines. Discussion will consider how this computing components can then be parts of a more complex chemical computing system, with a structure based on the membrane structure of τ-DPP systems, to move toward a wet implementation using the micro reactors technology. Our work thus exploits the close relation between such chemical processes and τ-DPP systems.

1 Introduction

In the recent years, several computational models derived from the formal abstraction of chemical reacting systems, such as the chemical abstract machine [3], and other inspired by the structure and functioning of living cells, have been proposed. One of these models, introduced in [14], is called P systems. The basic definition of P systems, also called membrane systems, consists of a hierarchical structure composed by several membranes. Inside every compartment, delimited by membranes, a set of evolution rules is placed. The rules (in particular, multiset rewriting rules) are used to describe the evolution of the objects occurring inside the system, which describe the state.

Among the different variants of P systems, here we consider τ-DPP, presented in [5]. Within the framework of τ-DPP, the probabilities are associated to the rules, following the method described by Gillespie in [7]. Moreover, τ-DPP extends the tau-leaping procedure [4] in order to quantitatively describe the behaviour of complex biological and chemical systems, embedded in membrane structures composed by different volumes.

This kind of chemical reacting systems, can be also implemented using micro reactors. Micro reactors [9], are laboratory devices composed of several reacting volumes (re-

actors) with a volume at the scale of the μl, connected by channels used to transport molecules.

The aim of this work is to show the correspondence between τ-DPP and chemical reacting systems occurring inside micro reactors. The close relation between the topological description of the two systems is clear, that is, both are composed by several volumes, and among these volumes it is possible to communicate molecules. Again, the approach based on multiset rewriting rules, that characterises τ-DPP, is similar to the chemical reacting process occurring within a micro reactor. Furthermore, both τ-DPP and chemical reacting systems emphasise the intrinsic stochasticity of chemical processes. The "noise" associated to the stochastic behaviour, rules the system dynamics at the micro-scales. At this scale, the small volumes and the high dilutions realize a system in which particles interaction should be described in a discrete fashion. Finally, the communication processes described by means of communication rules within τ-DPP, are strictly related to the channels interconnecting the reactors.

In this paper, we show how these analogies can be exploited to construct a wet implementation of P systems (in particular, of τ-DPP) and its feasibility. To obtain a description of τ-DPP that can be implemented using micro reactors, the encoding of boolean functions and register machine through chemical reactions, following the chemical computing principles, can be exploited.

Chemical computing [6] is a technique used to process information by means of real molecules, modified by chemical reactions, or using electronic devices programmed following principles taken from chemistry. Moreover, in chemical computing, the result of a computation is an emergent global behaviour based on the application of small systems characterised by chemical reactions.

Exploiting the definition of chemical network, the description of a system, as a set of reactions applied to a given set of molecular species, can be given. Moreover, with the chemical organisation theory, the set of (so called) organisations, within the set of molecular species, can be identified and then used to describe the behaviour of such system, as the movement between the organisations.

Note that, within the framework of chemical computing, every boolean function can be expressed by means of chemical reactions, hence, every problem can be encoded using a set of reactions.

A different kind of problem encoding will be then presented, this is based on the instructions of register machines [13]. This approach is similar to the one related to the chemical computing field. The idea is to use a set of chemical reactions to describe the instructions of the register machines. For instance, in the subsection 4.4, the description and the simulation of a decrement instruction, is presented.

The paper is organised as follows: in Section 2, membrane systems and its variant of

τ-DPP are explained. The chemical computing framework and chemical organisation theory are presented in Section 3. In Section 4, we show the results of the simulations of the small components used to describe bigger systems, such as XOR and NAND logic circuits, and the decrement instruction of a register machine. We conclude with some discussion in Section 5.

2 Membrane systems and τ-DPP

In this section we describe the framework of membrane systems, or P systems [15], recalling their basic notions and definitions.

We then present τ-DPP, a computational method firstly introduced in [5], used to describe and perform stochastic simulations of complex biological or chemical systems. The "complexity" of the systems managed by means of τ-DPP, is not only related to the number of the reactions (rules) and species (objects) involved, but it results from the topological structure of the system, that can be composed by many volumes. Generally speaking, the systems described using τ-DPP are represented through multiset rewriting rules placed inside the volumes forming the system structure.

2.1 Membrane systems P systems, or membrane systems, were introduced in [14] as a class of unconventional computing devices of distributed, parallel and nondeterministic type, inspired by the compartmental structure and the functioning of living cells.

In order to define a basic P system, three main parts need to be introduced: the *membrane structure*, the *objects* and the *rules*.

The *membrane structure* defines the framework where multisets of objects are placed, and evolve by means of evolution rules. Another feature of the membrane structure is related to the management of objects communication among different membranes, using particular evolution rules. The definition of membrane structure is given through a set of membranes with a distinct label (usually distinct numbers), hierarchically organized inside a unique membrane, named *skin membrane*. Among others, a representation of a membrane structure is given by using a string of square parentheses. Every pair of matching parentheses is placed inside a special pair, denoting the skin membrane. Hence, every pair characterises a membrane of the system.

For instance, the string $\mu = [_0\,[_1\,[_2\,[_3\,]_3\,]_2\,]_1\,[_4\,]_4\,]_0$, represents a membrane structure composed by 5 membranes, organised in four hierarchical levels. Moreover, the same membrane structure can be represented by the string $\mu' = [_0\,[_4\,]_4\,[_1\,[_2\,[_3\,]_3\,]_2\,]_1\,]_0$, hence, two pairs of matching parentheses, placed at the same hierarchical level, can be interchanged, together with their contents. This means that the order of pairs of parentheses is irrelevant, whereas their respective relationships are important.

Each membrane identifies a *region*, delimited by it and the adjacent membranes, possibly present inside it. The number of membranes in a membrane structure is called the *degree* of the P system. Finally, the whole space outside the skin membrane is called the *environment*.

The internal state of a P system is described by the *objects* occurring inside the membranes. An object can be either a symbol or a string over a specified and fixed finite alphabet V. In order to denote the presence of multiple copies of objects inside the membranes, multisets are usually used. A multiset consists of two components: the base set identifying the elements that constitute the multiset, and a function associating to each element its multiplicity in the multiset. In the framework of P systems, the multiset associated with membrane i is a map $M_i : V \to \mathbb{N}$.

The objects occurring inside the membranes of a P systems, specified with multisets M_i, are transformed by means of *evolution rules*. These rules are multiset rewriting rules of the form $r_i : u \to v$, where u and v are multisets of objects. The meaning of a rule is that the multiset u is modified into the multiset v; moreover, it is possible to associate to v the target of the rule, that is the membrane where the multiset is placed when the rule is applied. There are three different types of target. If the target is $here$, then the object remains in the region where the rule is executed. If the target is out, then the object is sent out from the membrane containing the rule and placed to the outer region. Note that, if a rule with this target indication is applied inside the skin membrane, then the object is sent to the environment. If the target is in_j, where j is a label of a membrane, then the object is sent into the membrane labelled with j. It is possible to apply this kind of rule, only if the membrane j is immediately inside the membrane where the rule is applied.

Starting from an initial configuration (described by a membrane structure containing a certain number of objects and a fixed set of rules), and letting the system evolve, a computation is obtained. A universal clock is assumed to exist: at each step, all rules in all regions are simultaneously applied to all objects which are the subjects of evolution rules. So doing, the rules are applied in a maximal parallel manner, hence the membranes evolve simultaneously. If no further rule can be applied, the computation halts. The result of a computation is the multiset of objects contained into an *output membrane* or emitted from the skin of the system to the environment.

For a complete and extensive overview of P systems, we refer the reader to [15], and to the P Systems Web Page (`http://ppage.psystems.eu`).

2.2 τ-DPP We now introduce a novel stochastic simulation technique called τ-DPP [5]. The aim of τ-DPP is to extend the single-volume algorithm of tau-leaping [4], in order to simulate multi-volume systems, where the distinct volumes are arranged according to a specified hierarchy. The structure of the system is required to be kept fixed during the evolution. In Section 2.1, we shown that the framework of membrane sys-

tem satisfies this requirement, hence, the spacial arrangement of P system is exploited in the description τ-DPP. In particular, there is a close correspondence between τ-DPP and a variant of P system called dynamical probabilistic P systems (DPP). DPP were introduced in [18]: they exploit the membrane structure of P systems but they associate probabilities with the rules, and such values vary (dynamically), according to a prescribed strategy, during the evolution of the system. For the formal definitions of DPP and examples of simulated systems, we refer the reader to [16, 17, 1, 2].

There is a difference between DPP and τ-DPP: the former provides only a qualitative description of the analysed system while the latter is able to give a quantitative description. Though, the need for an accurate *quantitative* tool, led to the definition of τ-DPP [5]. This approach is designed to share a common time increment among all the membranes, which allows to generate an accurate distribution of rules in each compartment. This improvement is achieved using, inside the membranes of τ-DPP, a modified tau-leaping algorithm, which gives the possibility to simulate the time evolution of every volume as well as that of the entire system.

The internal behaviour of the membranes, is therefore described by means of a modified tau-leaping procedure. The original method, first introduced in [8], is based on the stochastic simulation algorithm (SSA) presented in [7]. These approaches are used to describe the behaviour of chemical systems, computing the probabilities of the reactions placed inside the system and the length of the step (at each iteration), according to the current system state. While SSA is proved to be equivalent to the Chemical Master Equation (CME), therefore it provides the exact behaviour of the system, the tau-leaping method describes an approximated behaviour with respect to the CME, but it is faster for what concerns the computational time required.

As previously said, in τ-DPP we exploit a modified tau-leaping algorithm, in order to describer the correct behaviour of the whole system. This is achieved by letting all the volumes evolve in parallel, through a strategy used to compute the probabilities of the rules (and then, to select the rules that will be executed), and to choose the "common" time increment that will be used to update the system state.

The method applied for the selection of the length of the time step is the following. Each membrane independently computes a time increment, based on its internal state (exploiting the tau-leaping procedure). The smallest time increment is then selected and used to describe the evolution of the whole system, during the current iteration. Since all volumes *locally* evolve according to the same time increment, τ-DPP is able to correctly work out the *global* dynamics of the system. Moreover, using the "common" time increment inside the membranes, it is possible to manage the communication of objects among them. This is achieved because the volumes are naturally *synchronised* at the end of each iterative step, when all the rules are executed.

The modified tau-leaping procedure of τ-DPP is used to select the time increment that will govern the evolution of the system as well as the set of rules that will be executed

during the current leap. Furthermore, in order to describe the correct behaviour of the membranes and of the entire system, all the volumes have to independently select the kind of evolution they will follow. The membrane can evolve in three different manners (as described in [4]): executing either (1) a SSA-like step, or (2) non-critical reactions only, or (3) a set of non-critical reactions plus one critical reaction. A reaction is critical, if its reactants are present inside the system in very small amounts. The critical and non-critical reaction sets are identified at the beginning of every iteration. The separation of these two sets is needed in order to avoid the possibility of obtaining negative quantities after the execution of the rules (we refer the reader to [8] for more details).

After this first stage of the procedure, the membranes select the rules that will be used to update the system, exploiting the common time increment previously chosen. A detailed description of the algorithm will be given later on.

Formally, a τ-DPP Υ is defined as

$$\Upsilon = (V_1, \ldots, V_n, \mu, \mathcal{S}, M_1, \ldots, M_n, R_1, \ldots, R_n, C_1 \ldots C_n),$$

where:

- $V_1, \ldots, V_n$ are the volumes of the system, $n \in \mathbb{N}$;
- μ is a membrane structure representing the topological arrangement of the volumes;
- $\mathcal{S} = \{X_1, \ldots, X_m\}$ is the set of molecular species, $m \in \mathbb{N}$, that is, the alphabet of the system;
- $M_1, \ldots, M_n$, are the sets of multisets occurring inside the membranes $V_1, \ldots, V_n$, representing the internal state of the volumes. The multisets M_i $(1 \leq i \leq n)$ is defined over S^*;
- $R_1, \ldots, R_n$ are the sets of rules defined in volumes $V_1, \ldots, V_n$, respectively. A rule can be of internal or of communication type (as described below);
- $C_1, \ldots, C_n$ are the sets of stochastic constants associated to the rules defined in volumes $V_1, \ldots, V_n$.

Inside the volumes of a system described by means of τ-DPP, two kind of evolution rules can be placed. These rules are called *internal* and *communication* rules. Internal rules describe the evolution of objects that remain in the same region where the rule is executed. Communication rules, send objects from the membrane where they are applied to an adjacent volume. Moreover they can also modify the objects during the communication process.

The sets of stochastic constants $C_1, \ldots, C_n$, associated to the set of rules $R_1, \ldots, R_n$, are needed to compute the probabilities of the rule applications (also called propensity functions), along with a combinatorial function depending on the left-hand side of the rule [7].

The general form of internal and communication rules is $\alpha_1 S_1 + \alpha_2 S_2 + \cdots + \alpha_k S_k \rightarrow (\beta_1 S_1 + \beta_2 S_2 + \cdots + \beta_k S_k, target)$, where $S_1, \ldots, S_k \in \mathcal{S}$ are the objects involved in the rule and $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k \in \mathbb{N}$ are the coefficients associated to the objects. Note that we will usually consider the case where at most three objects appear in the left-hand side of the rule. This assumption is related to the fact that the probability of the combination of more than three objects is close to zero.
The target of the rules occurring inside the volumes of a τ-DPP can be one of the following:

- $here$: the objects are modified and remain in the volume where the rule is applied (internal rule)
- out: this means that the products of the rule are "sent outside" the source volume, where the rule is applied, to the adjacent outer volume;
- in_{label}: this means that the products of the rule are "sent inside" the volume with the $label$ specified in the target. This kind of rules are only allowed if the target volume is placed inside the source membrane, and the two volumes are adjacent (that is, there exists no other volume placed between the source and the target volume).
- in: this means that the products of the rule are nondeterministically sent to any of the volumes placed inside the source membrane. This kind of rule can be used instead of a set of rules with specific targets in_{label} (one rule for each inner volume).

Another difference between internal and communication rules is related to the time increment (τ) computation. In the procedure used to compute τ, while internal rule are involved using both left-hand and right-hand sides, in the case of communicating rules, the method considers only their left-hand side. This distinction is needed because the right-hand side of internal rule modifies the internal state of the membrane where the rule is applied whereas the right-hand side of communicating rule affects the state of another membrane, hence it is not considered during the τ computation.

Obviously, the right-hand side of communication rules will contribute to the update of the system state, which takes place at the end of the iteration step, and will be therefore considered to determine the state of the target volume for the next iteration step.

We now describe the τ-DPP algorithm needed to simulate the evolution of the entire system. Each step is executed *independently* and *in parallel* within each volume V_i ($i = 1, \ldots, n$) of the system. In the following description, the algorithm execution naturally proceeds according to the order of instructions, when not otherwise specified by means of "go to" commands.

Step 1. Initialisation: load the description of volume V_i, which consists of the initial quantities of all object types, the set of rules and their respective stochastic constants.

Step 2. Compute the propensity function a_μ of each rule r_μ, $\mu = 1, \ldots, m$, and evaluate the sum of all the propensity functions in V_i, $a_0 = \sum_{\mu=1}^{m} a_\mu$. If $a_0 = 0$, then go to *step 3*, otherwise go to *step 5*.

Step 3. Set τ_i, the length of the step increment in volume V_i, to ∞.

Step 4. Wait for the communication of the smallest time increment $\tau_{min} = \min\{\tau_1, \ldots, \tau_n\}$ among those generated independently inside all volumes $V_1, \ldots, V_n$, during the current iteration, then go to *step 13*.

Step 5. Generate the step size τ_i according to the internal state, and select the way to proceed in the current iteration (i.e. SSA-like evolution, or tau-leaping evolution with non-critical reactions only, or tau-leaping evolution with non-critical reactions and one critical reaction), using the selection procedure defined in [4].

Step 6. Wait for the communication of the smallest time increment $\tau_{min} = \min\{\tau_1, \ldots, \tau_n\}$ among those generated independently inside all volumes, during the current iteration. Then:

- if the evolution is SSA-like and the value $\tau_i = \tau_{SSA}$ generated inside the volume is greater than τ_{min}, then go to *step 7*;
- if the evolution is SSA-like and the value $\tau_i = \tau_{SSA}$ is equal to τ_{min}, then go to *step 10*;
- if the evolution is tau-leaping with non-critical reactions plus one critical reaction, and the value $\tau_i = \tau_{nc1c}$ is equal to τ_{min}, then go to *step 11*;
- if the evolution is tau-leaping with non-critical reactions plus one critical reaction, and the value $\tau_i = \tau_{nc1c}$ is greater than τ_{min}, then go to *step 12*;
- if the evolution is tau-leaping with non-critical reactions only ($\tau_i = \tau_{nc}$), then go to *step 12*.

Step 7. Compute $\tau_{SSA} = \tau_{SSA} - \tau_{min}$.

Step 8. Wait for possible communication of objects from other volumes, by means of communication rules. If some object is received, then go back to *step 2*, otherwise go to *step 9*.

Step 9. Set $\tau_i = \tau_{SSA}$ for the next iteration, then go back to *step 6*.

Step 10. Using the SSA strategy [7], extract the rule that will be applied in the current iteration, then go to *step 13*.

Step 11. Extract the critical rule that will be applied in the current iteration.

Step 12. Extract the set of non-critical rules that will be applied in the current iteration.

Step 13. Update the internal state by applying the extracted rules (both internal and communication) to modify the current number of objects, and then check for objects (possibly) received from the other volumes. Then go back to *step 2*.

The algorithm begins loading the initial conditions of the membrane. The following operation is the computation of the propensity functions (and the sum of them) in order to check if, inside the membrane, it is possible to execute some reaction. If the sum of the propensity functions is zero, then the value of τ is set to ∞ and the membrane waits for the communication of the smallest τ computed among the other membranes (τ_{min}) in order to synchronise with them and then checks if it is the target of some

communication rule applied inside the other volumes. These operations are needed in order to properly update the internal state of the membrane.

However, if the sum of all the propensity functions is greater than zero, the membrane will compute a τ value based only on its internal state, following the first part of the original tau-leaping procedure [4]. Besides this operation, the membrane selects the kind of evolution for the current iteration (like the computation of τ, this procedure is executed independently from the other volumes).

The algorithm proceeds to *Step 6*, where the membrane receives the smallest τ value computed by the volumes. This will be the common value used to update the state of the entire system. It is necessary to proceed inside every membrane using the same time increment, in order to manage the communication of objects.

At this stage, the membrane knows the length of the time step and the kind of evolution to perform. The next step consists in the extraction of the rules that will be applied in the current iteration. In order to properly extract the rules, several conditions need to be checked.

In the case the membrane is evolving using the SSA strategy: if τ_{min} is the value generated inside of it, then it is possible to extract the rule, otherwise the execution of the rule is not allowed, because the step is "too short". In the next stage, the membrane verifies for possible incoming objects, to update its internal state according to the communication rules (possibly) executed inside the other regions. Finally, if its state is changed (according to some internal or communication rule), then the membrane, in the successive iteration, will compute a new value of τ. On the contrary, the value of the time increment used, will be the result of the application of *Step 7* of the algorithm.

If the evolution strategy corresponds to a tau-leaping step with the application of a set of non-critical reactions and one critical reaction, the algorithm verifies if the value of τ computed by the membrane is equal to τ_{min}. If this is true, the membrane selects the set of non-critical reactions to execute as well as the critical reaction. The execution of the critical reaction is allowed because, here τ_{min} represents the time needed to execute it. On the other hand, the application of the critical reaction is forbidden and the membrane will execute non-critical reactions only.

If the membrane is following the tau-leaping strategy with the execution of non-critical reactions only, τ_{min} is used to extract the rules (belonging to the set of non-critical) to apply in the current iteration.

The last step is the system update. Here every membrane executes the selected rules and update its state according to both internal and communication rules. This step is executed in parallel inside every membrane, therefore it is possible to correctly manage the passage of objects and to synchronise the volumes.

3 Chemical computing

In this section we introduce the basic notions of chemical computing systems and chemical organisation theory, showing how to exploit them, together with membrane systems, in order to obtain a representation suitable for the wet implementation of the problem in analysis, using micro reactors.

Biological systems are characterised by different mechanisms, employed in their evolution, that are able to process information. This methods are: robust, self-organising, adaptable, decentralised, asynchronous, fault-tolerant and evolvable. The global information process comes from the application, inside the biological system, of a large number of simple components. In particular, information is transformed by means of chemical processes, and for this reason, chemical reactions have been used to build a novel computational paradigm [6]. This new approach is called chemical computing, and it is related to the computation with real molecules as well as the electronic devices, programmed using principles taken from chemistry

In general, the analysis of the solutions of chemical reaction processes, is hard because of its nonlinearity. The same problems are related to the analysis of biological systems since the behaviour of local parts can be very different from the global behaviour.

In order to work out this problem, the notions of chemical organisation theory can be used to obtain the emergent behaviour of the system, starting from its small components, hence linking the evolution governed by every single reaction rule with the global dynamics of the system.

Chemical organisation theory is used to identify a hierarchy of self maintaining sub-networks, belonging to a chemical reaction network. These sub-networks are called organisations. In particular, a chemical organisation is a set of molecular species that is algebraically closed and stoichiometrically self-maintaining. To define the organisations and the properties of this particular type of sub-network, we first need to introduce reaction networks.

A *reaction network* is a tuple $\langle \mathcal{M}, \mathcal{R} \rangle$, where $\mathcal{M}$ is a set of molecular species and $\mathcal{R}$ is a set of reactions (also called rules). The rules in $\mathcal{R}$ are given by the relation $\mathcal{R} : P_{\mathcal{M}}(\mathcal{M}) \times P_{\mathcal{M}}(\mathcal{M})$ where $P_{\mathcal{M}}(\mathcal{M})$ denotes the set of all the multisets of the elements in $\mathcal{M}$. The general form of a reaction is $\alpha_1 m_1 + \alpha_2 m_2 + \cdots + \alpha_k m_k \rightarrow \beta_1 m_1 + \beta_2 m_2 + \cdots + \beta_k m_k$, where $m_1, \ldots, m_k \in \mathcal{M}$ are the molecular species involved in the rule and $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k \in \mathbb{N}$ are the coefficients associated to the molecules.

As previously said, a set of molecular species belonging to $\mathcal{R}$ is called organisation, if the properties of closure and self-maintenance are satisfied. Here we report an informal definition of these properties, we refer the reader to [12], for formal definitions and further details.

A set of molecular species $\mathcal{C} \in \mathcal{R}$ is *closed*, if its elements are involved in reactions that produce only molecular species of the set $\mathcal{C}$. On the other hand, the *self-maintenance* property is satisfied when the molecules consumed by the reactions involved in the set, can also be produced by some other rule related to the self-maintaining set. Note that, in order to find the organisations of a chemical network, only stoichiometric information (set of rules) is needed.

The set of organisation of a chemical network, can be exploited to describe the dynamics of the system, by means of the movement among its elements. In particular, only the algebraic analysis of chemical organisation is needed in order to obtain the behaviour of the system. This analysis consists in the study of processes where molecular species appear or disappear from the system (that is, when their amount become positive or go to zero). Furthermore, the behaviour of the system, as described above, can either take place spontaneously or can be induced by means of external events, such as the addition of input molecules.

If we want to use these reaction networks to compute, we will assume to describe a computational problem by means of a boolean function, which in turn can be computed as a composition of many simple functions, such as the binary NAND. Therefore, we will create a reaction network (called boolean network), based on a set of boolean functions and boolean variables.

We define a boolean network using a set of M boolean functions and a set of N (with $N \geq M$) boolean variables $\{b_1, \ldots, b_M, \ldots, b_N\}$. The variables b_j, such that $1 \leq j \leq M$, are determined by the boolean functions (they are also called internal variables). The remaining variables (b_j such that $M < j \leq N$) represent the input variables of the boolean network. The values computed by the set of N boolean functions, are defined as $\{b_i = F_i(b_{q(i,1)}, \ldots, b_{q(i,n_i)})$ with $i = 1, \ldots, M\}$. $b_{q(i,k)}$ is the value of the boolean variable corresponding to the k-th argument of the i-th function. In general, the function F_i has n_i arguments, therefore, there are 2^{n_i} different input combinations. The truth table T_i of the function F_i is then composed by 2^{n_i} rows and $n + 1$ columns, where the first n columns contain values for the arguments of the function and the last one is the corresponding output.

Given a boolean network (as described above), the associated reaction network $\langle \mathcal{M}, \mathcal{R} \rangle$ is defined ad follows. For each boolean variable b_j, we add two different molecular species to $\mathcal{M}$, representing the values 0 and 1 of the variable. In particular, lowercase letters are used for the molecular species representing the value 0 and uppercase letters for the value 1 of the variables. Therefore, the set $\mathcal{M}$ contains $2N$ molecular species. The set $\mathcal{R}$ of rules, is composed by two kinds of reactions: *logical* and *destructive*. Logical reactions are related to the rows of the truth tables of the functions involved in the boolean network; hence the left-hand side of the rule represents the input values of the boolean function, while the right-hand side is the output value. The destructive reactions are needed to avoid the possibility to have, inside the system, two molecular species representing both states of the same variable at the same time (i.e. two molecules

representing the state 0 and 1 of the same boolean variable). In this case, the state of the variable becomes undefined, and a rule that degrades the two corresponding molecular species, is required.

The resulting chemical network $\langle \mathcal{M}, \mathcal{R} \rangle$ implements the boolean network without inputs specified. The input variables of the boolean network must be externally initialised because they are not set by the boolean functions. The initialisation is encoded by means of inflow reactions. These reactions are zero-order reactions producing substances from the empty set.

4 Building and simulating *component reaction networks* in τ-DPP

To lay out our path from a model of computation to a chemical computing device, we build and simulate small τ-DPP systems using techniques inspired by the literature on reaction systems [6, 12]. Those small systems must be powerful enough to compute, when assembled in more complex systems, any computable (boolean) function.

Hence, in this section we describe the implementations of the NAND and XOR logic circuits, and of the decrement and increment instructions of register machines, through sets of τ-DPP chemical reactions. We then present some simulation results of our systems.

We recall that *register machines* are universal abstract computing devices, where a finite set of uniquely labelled instructions is given, and which at any time keep updated a finite set of registers (holding integer numbers) by performing a sequence of instructions, chosen according to their labels. Every instruction can be of one of the following types, here informally introduced:

- `ADD`: a specified register is increased by 1, and the label of next instruction is chosen nondeterministically between two labels specified in the instruction,
- `SUB`: a specified register is checked, and if it is non-empty it is decreased by 1, otherwise it will not be changed; the next label will be differently chosen in the two cases,
- `HALT`: the machine stops.

Later, we will describe τ-DPP implementations of `SUB` and `ADD` instructions.

4.3 The NAND and XOR logic circuits The NAND logic circuit has been implemented with the sequential composition of an AND and a NOT gate as shown in Figure 4.1. Following the chemical computing guidelines described in Section 3, we defined the logic circuits with the rules listed in Table 4.1. The constant values reported in the table have been used to perform the simulation by means of τ-DPP. Note that the rules $r_{11}, \ldots, r_{14}$ represent the inputs of the gate. For instance, when the constants of the

rules r_{11} and r_{13} are set to 1, the input given to the NAND gate is 0 for both the input lines. The rationale behind this, is that the different inputs for the system are obtained producing the corresponding molecular species.

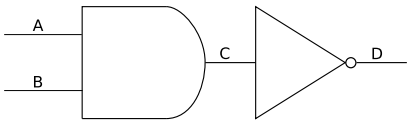


Fig. 4.1 The NAND logic gate

Table 4.1 Reaction rules for the NAND unit. The initial multiset is set to 0 for all the molecular species.

Reaction Rule		**Constant**
r_1	$a + b \to c$	$1 \cdot 10^{-3}$
r_2	$a + B \to c$	$1 \cdot 10^{-3}$
r_3	$A + b \to c$	$1 \cdot 10^{-3}$
r_4	$A + B \to C$	$1 \cdot 10^{-3}$
r_5	$c \to D$	$1 \cdot 10^{-2}$
r_6	$C \to d$	$1 \cdot 10^{-2}$
r_7	$a + A \to \lambda$	$1 \cdot 10^{-1}$
r_8	$b + B \to \lambda$	$1 \cdot 10^{-1}$
r_9	$c + C \to \lambda$	$1 \cdot 10^{-1}$
r_{10}	$d + D \to \lambda$	$1 \cdot 10^{-1}$
r_{11}	$\lambda \to a$	$c_{11} \in \{1, 0\}$
r_{12}	$\lambda \to A$	$c_{12} \in \{1, 0\}$
r_{13}	$\lambda \to b$	$c_{13} \in \{1, 0\}$
r_{14}	$\lambda \to B$	$c_{14} \in \{1, 0\}$

Rules $r_1, \ldots, r_4$ compute AND, rules $r_5, \ldots, r_6$ compute NOT, rules $r_7, \ldots, r_{10}$ will clean the system when going to change its input and, in general, when both values for a variable are present in the system.

Exploiting the set of rules for the NAND gate, it is then possible to define the τ-DPP which encodes the logic circuit. Formally, the τ-DPP Υ_{NAND} is defined as

$$\Upsilon_{NAND} = (V_0, \mu, \mathcal{S}, M_0, R_0, C_0),$$

where:

- V_0 is the unique volume of the NAND unit;
- μ is the plain membrane structure $[_0\ ]_0$;

- $\mathcal{S} = \{a, A, b, B, c, C\}$ is the set of molecular species;
- $M_0 = \{a^{m_a}, A^{m_A}, b^{m_b}, B^{m_B}, c^{m_c}, C^{m_C}, d^{m_d}, D^{m_D}\}$, is the set of multisets occurring inside the membrane V_0;
- $R_0 = \{r_1, \ldots, r_{14}\}$ is the set of rules defined in volumes V_0 and reported for clarity in Table 4.1. Due to the flat membrane structure, all the rules here involved are internal.
- $C_0 = \{c_1, \ldots, c_{14}\}$ is the sets of stochastic constants associated to the rules defined in V_0, reported for clarity in Table 4.1.

In Figure 4.2, the results of the simulation of the NAND gate are reported. In the initial configuration of the system, the multisets are empty, that is, the amounts of all the molecular species are set to zero. At time $t = 0$, the input of the system is a, B, corresponding to the first input line set to zero and the second line set to one. This corresponds to a τ-DPP configuration where the constants of rules r_{11} and r_{14} are set to 1, while the constants of rules r_{12} and r_{13} are set to zero. The output obtained with this configuration is 1, indeed the system, at the beginning of the simulation, produces the molecules D corresponding to the expected output value. At time $t = 400$ the input values of the system change from a, B to A, B. The system starts producing d molecules, but the output of the system changes only when all the D molecules have been degraded (by means of rule r_{10}) and the molecules d are then accumulated inside the membrane.

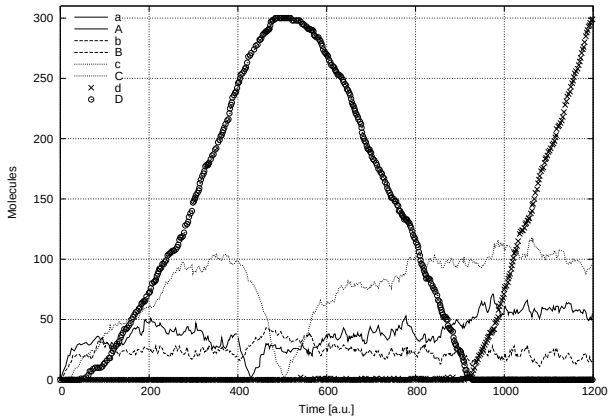


Fig. 4.2 Plot of the dynamics of the NAND unit with two inputs in succession. The initial multiset is set to 0 for all the molecular species.

The XOR gate (see Figure 4.3) has been implemented using the set of rules listed in Table 4.2. The constant values reported in the table have been used to perform the simulation by means of τ-DPP. The rules $r_8, \ldots, r_{11}$ represent the inputs of the gate. For instance, when the constants of the rules r_8 and r_{10} are set to 1, the input given to the

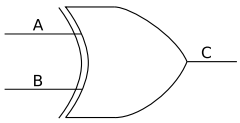


Fig. 4.3 The XOR logic gate

XOR gate is 0 for both the input lines. The rational behind this, is that the different inputs for the system are obtained producing the corresponding molecular species.

Table 4.2 Reaction rules for the XOR unit. The initial multiset is set to 0 for all the molecular species.

Reaction Rule		**Constant**
r_1	$a + b \to c$	$1\ 10^{-3}$
r_2	$a + B \to C$	$1\ 10^{-3}$
r_3	$A + b \to C$	$1\ 10^{-3}$
r_4	$A + B \to c$	$1\ 10^{-3}$
r_5	$a + A \to \lambda$	$1\ 10^{-1}$
r_6	$b + B \to \lambda$	$1\ 10^{-1}$
r_7	$c + C \to \lambda$	$1\ 10^{-1}$
r_8	$\lambda \to a$	$c_8 \in \{1, 0\}$
r_9	$\lambda \to A$	$c_9 \in \{1, 0\}$
r_{10}	$\lambda \to b$	$c_{10} \in \{1, 0\}$
r_{11}	$\lambda \to B$	$c_{11} \in \{1, 0\}$

Formally, the τ-DPP Υ_{XOR}, corresponding to the XOR logic circuit, is defined as

$$\Upsilon_{XOR} = (V_0, \mu, \mathcal{S}, M_0, R_0, C_0),$$

where:

- V_0 is the unique volume of the XOR unit;
- μ is the plain membrane structure $[\ ]_0$;
- $\mathcal{S} = \{a, A, b, B, c, C, d, D\}$ is the set of molecular species;
- $M_0 = \{a^{m_a}, A^{m_A}, b^{m_b}, B^{m_B}, c^{m_c}, C^{m_C}\}$, is the set of multisets occurring inside the membrane V_0;
- $R_0 = \{r_1, \ldots, r_{11}\}$ is the set of rules defined in volumes V_0 and reported for clarity in Table 4.2. Due to the flat membrane structure, all the rules here involved are internal.
- $C_0 = \{c_1, \ldots, c_{11}\}$ is the sets of stochastic constants associated to the rules defined in V_0 reported for clarity in Table 4.2.

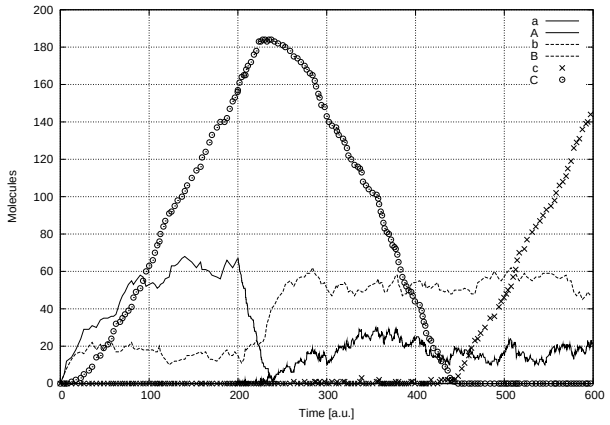


Fig. 4.4 Plot of the dynamics of the XOR unit with two input in succession. The initial multiset is set to 0 for all the molecular species.

In Figure 4.4, the results of the simulation of the XOR gate are reported. In the initial configuration of the system, the multisets are empty, that is, the amounts of all the molecular species are set to zero. At time $t = 0$, the input of the system is a, B, corresponding to the first input line set to zero and the second one set to one. This corresponds to a τ-DPP configuration where the constants of rules r_8 and r_{11} are set to 1, while the constants of rules r_9 and r_{10} are set to zero. The output obtained with this configuration is 1, indeed the system, at the beginning of the simulation, produces the molecules C corresponding to the expected output value. At time $t = 200$ the input values of the system change from a, B to A, B. The system starts producing c molecules, but the output of the system changes only when all the C molecules have been degraded (by means of rule r_7) and the molecules c are then accumulated inside the membrane.

4.4 The SUB instruction We now describe and simulate a 2-membranes τ-DPP system reproducing, by means of chemical reactions operating on a set of molecular species, the behaviour of a SUB instruction of a register machine. This type of instruction is shown first because it hides a conditional behaviour, checking whether a register is zero or not, and respectively choosing a different label for the next instruction, and the availability of conditional instructions is a key issue in computing devices.

We consider a SUB instruction checking a register whose value is associated to the presence of objects u in volume V_1, and the resulting choice for the next label will be represented by the appearing in volume V_0 either of objects p or z. Finally, the triggering of this instruction will be given by the injection in V_0 of a quantity of objects s.

Formally, a τ-DPP Υ_{SUB} is defined as

$$\Upsilon_{SUB} = (V_0, V_1, \mu, \mathcal{S}, M_0, M_1, R_0, R_1, C_0, C_1),$$

where:

- V_0, V_1 are the volumes of the SUB unit;
- μ is the nested membrane structure $[_0\,[_1\ ]_1\,]_0$;
- $\mathcal{S} = \{p, s, s', u, z, z'\}$ is the set of molecular species;
- $M_0 = \{p^{m_p}, s^{m_s}, s'^{m_{s'}}, z^{m_z}\},\ M_1 = \{p^{m_p}, s^{m_s}, u^{m_u}, z^{m_z}, z'^{m_{z'}}\}$, are the multisets occurring inside the membranes V_0 and V_1 respectively;
- $R_0 = \{r_1, \ldots, r_5\},\ R_1 = \{r_1, \ldots, r_3\}$ are the sets of rules defined in volumes V_0, V_1 respectively, and reported for clarity in Table 4.3;
- $C_0 = \{c_1, \ldots, c_5\},\ C_1 = \{c_1, \ldots, c_3\}$ is the sets of stochastic constants associated to the rules defined in V_0, V_1 reported for clarity in Table 4.3.

Table 4.3 Reaction rules for the SUB unit (R_0 on the left and R_1 on the right). The initial multisets are $\{s'^{40}\}$ in V_0, and $\{u^{20}, z^5\}$ in V_1.

	Reaction Rule	**Constant**
r_1	$2p \to (p, here)$	1
r_2	$z + p \to (z, here)$	1
r_3	$2z \to (z, here)$	1
r_4	$s \to (s, in_0)$	1
r_5	$s' \to (s, here)$	$6\ 10^{-2}$

	Reaction Rule	**Constant**
r_1	$s + u \to (p, out)$	$1\ 10^3$
r_2	$s + z \to (z + z', here)$	1
r_3	$z' \to (z, out)$	1

This system is initialised with small quantities for molecular species, and this makes it fragile with respect to the inherent stochasticity, but our goal is to qualitatively show the required sharp change of behaviour occurring when the simulated register goes to zero.

The simulation starts with a positive register value within V_1, and it receives a sequence of SUB requests (rules r_5 and r_4), bounded in this example by the initial availability of s' molecules. Figure 4.5 shows the correct two phases of execution: in the first phase the counter is decremented and objects p are produced in V_0, but when the simulated counter reaches zero, only objects z will be produced.

4.5 The SUBADD module The SUB unit can be extended to be able to perform both a SUB and an ADD instruction, according to which objects it receives from outside: s or a, respectively. Rules are defined for the two possible operations, and the choice of objects avoid mixing them. The results can be seen in figures 4.6 and 4.7.

Formally, a τ-DPP Υ_{SUBADD} is defined as

$$\Upsilon_{SUBADD} = (V_0, V_1, V_2, \mu, \mathcal{S}, M_0, M_1, M_2, R_0, R_1, R_2, C_0, C_1, C_2),$$

where:

- V_0, V_1, V_2 are the volumes of the SUBADD module;

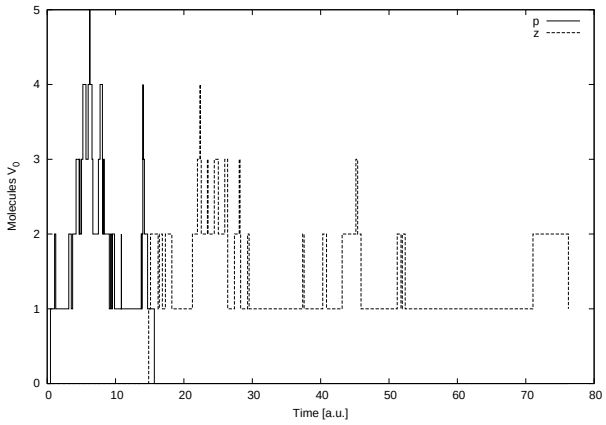


Fig. 4.5 Plot of the dynamics of the SUB unit.

- μ is the nested membrane structure $[_0\,[_1\,[_2\,]_2\,]_1\,]_0$;
- $\mathcal{S} = \{l, l', m, k, k', o, q, s, p, z, a, A, u, z', a'\}$ is the set of molecular species;
- $M_0 = \{l^{m_l}, l'^{m_{l'}}, s^{m_s}, z^{m_z}, p^{m_p}, k^{m_k}, k'^{m_{k'}}, A^{m_A}, o^{m_o}, m^{m_m}\}$, $M_1 = \{s^{m_s}, p^{m_p}, z^{m_z}, a^{m_a}, A^{m_A}\}$ and $M_2 = \{s^{m_s}, u^{m_u}, p^{m_p}, z^{m_z}, z'^{m_{z'}}, a'^{m_{a'}}\}$, are the multisets occurring inside the membranes V_0, V_1 and V_2 respectively;
- $R_0 = \{r_1, \ldots, r_8\}$, $R_1 = \{r_1, \ldots, r_8\}$, $R_2 = \{r_1, \ldots, r_5\}$ are the sets of rules defined in volumes V_0, V_1 and V_2 respectively, and reported for clarity in Table 4.4;
- $C_0 = \{c_1, \ldots, c_8\}$, $C_1 = \{c_1, \ldots, c_8\}$, $C_2 = \{c_1, \ldots, c_5\}$ is the sets of stochastic constants associated to the rules defined in V_0, V_1 and V_2 reported for clarity in Table 4.4.

5 Complete systems, discussion and open problems

Our results are a starting point, since they only tackle the building of basic elements of a computing device. A more complex problem is related to the connectivity among these components.

The general instance of boolean network, but also the general reaction network considered in literature, do require a complex grid of channels communicating variables/objects to the required destination gates/volumes.

For usual P systems, such a grid of channels can only reproduce a tree-like structure of nested membranes, communicating only between adjacent ones. We could move our studies to other models for P systems, which allow more free adjacency relations between membranes, such as “Tissue P Systems” [11]. But we can make two positive remarks: experimental chemical tools like micro reactors can be built reproducing any

Table 4.4 Reaction rules for the SUBADD module (R_0 on the left, R_1 on the right and R_2 on the bottom). The initial multisets M_0 and M_1 are empty, while M_2 is $\{u^{30}\}$.

	Reaction Rule	**Constant**
r_1	$l \to (l' + s, here)$	1
r_2	$s \to (s, in_1)$	1
r_3	$l' + z \to (m, here)$	1
r_4	$l' + p \to (m, here)$	1
r_5	$k \to (k' + a, in_1)$	1
r_6	$a \to (a, in_1)$	1
r_7	$k' + A \to (o, here)$	1
r_8	$k' + A \to (q, here)$	1

	Reaction Rule	**Constant**
r_1	$s \to (s, in_2)$	1
r_2	$2p \to (p, here)$	1
r_3	$2z \to (z, here)$	1
r_4	$p + z \to (p, here)$	1
r_5	$z \to (z, out)$	1
r_6	$p \to (p, out)$	1
r_7	$a \to (a, in_2)$	1
r_8	$A \to (A, out)$	1

	Reaction Rule	**Constant**
r_1	$s + u \to (p, out)$	$1\ 10^3$
r_2	$s + z \to (z + z', here)$	1
r_3	$z' \to (z, out)$	1
r_4	$a + u \to (2u + a', here)$	1
r_5	$a' \to (A, out)$	1

given grid of channels, while on the other hand the tree-like structure of P-systems does not rule out their universality [15].

Within our approach, by using SUBADD modules, we can outline the structure of a τ-DPP system simulating a complete register machine with just three levels of nested membranes: the skin membrane, and inside it a number of "register" membranes structured like volume V_1 in SUBADD module. The key idea to be developed, is to simulate the steps of the register machine by having in the skin membrane molecules representing the current instruction label. Then, for instance, if instruction l increments register r, then rules would be defined which produce objects a_r and send them to internal membrane representing register r. Continuing the example, that internal membrane will produce objects A_r, and a rule in skin membrane would transform pairs of objects $l + A_r$ into (non-deterministically chosen) objects m, where m is one of the outcome labels specified by the `ADD` instruction being simulated. (Other details to be specified will be those related to the halting of the computation.)

All this leads to some open problems worth being studied. The passage from single simple components to complete universal devices, with the required connectivity, how does it scale? It is well known that small universal register machines can be built [10], but their τ-DPP implementation, and eventually their chemical system implementation has to be evaluated. Moreover, the computational efficiency of these systems can be stud-

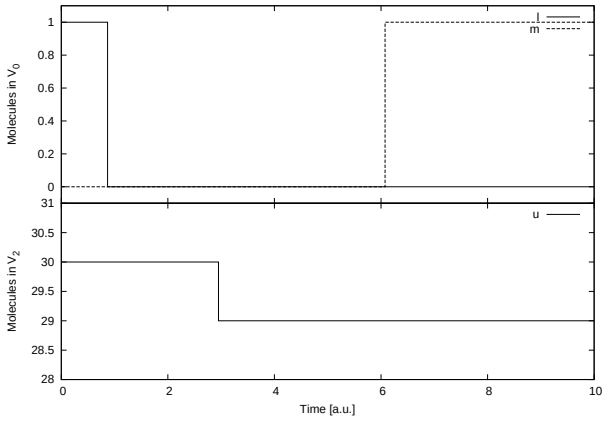


Fig. 4.6 Plot of the dynamics of the SUBADD module performing a decrement instruction on a register.

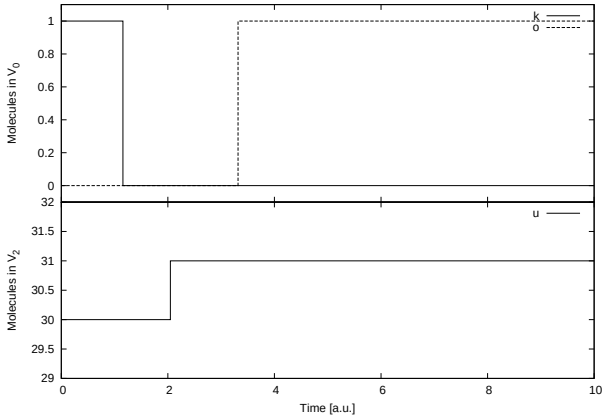


Fig. 4.7 Plot of the dynamics of the SUBADD module performing a increment instruction on a register.

ied, for instance with respect to NP-complete problems such as SAT. Anyway, the usual trade-off between space and time in structural complexity, perhaps has to be applied with negative results to τ-DPP, since objects could exponentially grow in polynomial time (by using rules like $p \rightarrow 2p$, but the space structure of volumes is fixed, and on the other hand the stochastic nature of the model substitute non-determinism by equiprobability, all motivated by chemical plausibility.

Bibliography

[1] D. Besozzi, P. Cazzaniga, D. Pescini, G. Mauri. Modelling metapopulations with stochastic membrane systems. *Biosystems*, 91:499–514, 2008. doi:10.1016/j.biosystems.2006.12.011 .

[2] D. Besozzi, P. Cazzaniga, D. Pescini, G. Mauri, Seasonal variance in P system models for metapopulations. *Progress in Natural Science*, 17:392–400, 2007.

[3] G. Berry, G. Boudol, The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[4] Y. Cao, D. T. Gillespie, and L.R. Petzold. Efficient step size selection for the tau-leaping simulation method. *Journal Chemical Physics*, 124:044109, 2006.

[5] P. Cazzaniga, D. Pescini, D. Besozzi, G. Mauri. Tau leaping stochastic simulation method in P Systems. 7th International Workshop, WMC 2006 (H.J. Hoogeboom, G. Păun, G. Rozenberg, A. Salomaa, eds.) *Lecture Notes in Computer Science*, Springer, 4361:298–313, 2006. doi:10.1007/11963516_19.

[6] P. Dittrich. Chemical computing. Unconventional Programming Paradigms (UPP 2004) *Lecture Notes in Computer Science*, 3566:19–32, 2005.

[7] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal Physical Chemistry*, 81:2340–2361, 1977.

[8] D. T. Gillespie and L.R. Petzold. Approximate accelerated stochastic simulation of chemically reacting systems. *Journal Chemical Physics*, 115:1716–1733, 2001.

[9] S. J. Haswell, R. J. Middleton, B. O'Sullivan, V. Skelton, P. Watts and P. Styring. The application of micro reactors to synthetic chemistry. *Chemical Communication*, 391–398, 2001. doi:10.1039/b008496o.

[10] I. Korec. Small universal register machines, *Theoretical Computer Science*, 168:267–301, 1996

[11] C. Martín-Vide, G. Păun, J. Pazos, A. Rodríguez-Patón. Tissue P systems, *Theoretical Computer Science*, 2:295–326, 2003

[12] N. Matsumaru, F. Centler, P Speroni di Fenizio, P. Dittrich. Chemical organization theory as a theoretical base for chemical computing. *International Journal of Unconventional Computing*, 3:285–309, 2005

[13] M. L. Minsky. Computation: finite and infinite machines. *Prentice-Hall*, Englewood Cliffs, 1967.

[14] G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143, 2000.

[15] G. Păun. *Membrane computing. an introduction*. Springer–Verlag, 2002. Berlin.

[16] D. Pescini, D. Besozzi, G. Mauri, and C. Zandron. Dynamical probabilistic P systems. *International Journal of Foundations of Computer Science*, 17:183–204, 2006.

[17] D. Pescini, D. Besozzi, and G. Mauri. Investigating local evolutions in dynamical probabilistic P systems. *Proceedings of Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'05)*. IEEE Computer Press, 440–447, 2005.

[18] D. Pescini, D. Besozzi, G. Mauri, and C. Zandron. Analysis and simulation

of dynamics in probabilistic P systems. In A. Carbone, N. Pierce, Eds., *DNA Computing, 11th International Workshop on DNA Computing, DNA11*, London, ON, Canada, June 6-9, 2005. Lecture Notes in Computer Science 3892, 236–247, Springer–Verlag, 2006.

Defining and Executing P-systems with Structured Data in K

Traian Serbănuță, Gheorghe Stefănescu, Grigore Rosu

University of Illinois at Urbana-Champaign, Department of Computer Science,
201 N. Goodwin, Urbana, IL 61801, USA
{tserban2,stefanes,grosu}@cs.uiuc.edu

K is a rewrite-based framework proposed for giving formal executable semantics to programming languages and/or calculi. K departs from other rewrite-based frameworks in two respects: (1) it assumes multisets and lists as builtin, the former modeling parallel features, while the latter sequential ones; and (2) the parallel application of rewriting rules is extended from non-overlapping rules to rules which may overlap, but on parts which are not changed by these rules (may overlap on "read only" parts). This paper shows how P-systems and variants can be defined as K (rewrite) systems. This is the first representation of P-systems into a rewrite-based framework that captures the behavior (reaction steps) of the original P-system step-for-step. In addition to providing a formal executable semantic framework for P-systems, the embedding of P-systems as K systems also serves as a basis for experimenting with and developing new extensions of P-systems, for example with structured data. A Maude-based application for executing P-systems defined in K has been implemented and experimented with; initial results show computational advantages of using structured objects in P-systems.

1 Introduction

K [23] (see also [14]) is a rewrite-based framework which has been proposed and developed (starting with 2003) as an alternative (to structural operational semantics) formal executable semantic framework for defining programming languages, language-related features such as type systems, and/or calculi. K's strength can be best reflected when defining concurrent languages or calculi, because it gives those a truly concurrent semantics, that is, one in which concurrent steps take place concurrently also in the semantics (instead of interleaving them, as conventional operational semantics do). K assumes as builtin and is optimized for multisets and lists, the former modeling parallel features, while the latter sequential ones. More importantly, rewriting rules in K can be applied concurrently even when they overlap, assuming that they do not change the overlapped portion of the term (may overlap on "read only" parts). Core parts of many programming languages or computation models are already defined in K, including Scheme [16],

KOOL [13], Milner's EXP language [17], Turing machines, CCS [18], as well as type systems for these, etc. - see [23].

A fast developing class of computation models was introduced by Paun in 1998 [20] exploiting ideas from chemistry and biology (see [22] for a recent survey). They are called *membrane systems* (or *P-systems*) and combine nested membrane structures with computation mechanisms inspired by the activity of living cells. There is a large variety of P-systems studied in the literature and a few toy implementations for developing applications. The original motivation was to link this research to formal language theory studies, but the model is more general, coming with important suggestions in many fields, for instance in the design of new parallel programming languages.

Both K definitions and P-systems use potentially nested membranes as a spatial modularization mechanism to encapsulate behaviors and to structure the systems (see Fig. 3.1(a) for an example of nested membranes). P-systems are inspired by chemistry and biology, using "objects" (abstract representations of chemical molecules) which interact; all communication is at the object level. Objects move from region to region (in the basic model, these are neighboring regions) either directly, or by using symport/antiport mechanisms. When far reaching regions are targeted, special tags are to be added to objects to reach the destination and, in most of the models, the objects have to travel through membranes from region to region to reach the final destination.

The K-framework uses a similar membrane structure (called "cell"), but for a different goal, leading to important differences. The main objective of K is to model high-level programming languages and calculi, for instance allowing OO- or multi-threading programming. To this end, the objects within the membranes are structured. This structure is both in space and in time. The spatial aspect refers to the use of algebraic terms to describe the objects floating in the membrane soups (regions). Due to this algebraic structure, one has more power to express object interaction. However, there is another equally important mechanism, which is not explicitly present in P-systems or in CHAMs [6, 7], namely the use of computation tasks, as control structures evolving in time. To this end, a new data type is builtin in the K framework, the *list* structure,[14] used to capture sequential orders on tasks' execution. The "communication" in K is at a high level, data being "moved" from a place to another place in a single step. It is this combination of structured data and their use in a mixture of nested soups (for parallel processes) and lists (for sequential tasks) which makes it possible to effectively define various high-level languages in K.

P-systems may be described without membranes. Indeed, using the tree associated to the membrane structure, one can tag each object with the path in the tree from the root to the membrane where the object is in. Now, the objects may be used in a unique huge soup and the evolution rules, previously used in specific regions, become global

[14] Lists can be encoded with (multi)sets, but allowing them as "first-class citizens" gives the K user the capability to match fragments of lists.

rules, but applied to similar path-tagged objects. This observation highlights the advantages of using membrane systems: the matching and the interaction between objects are localized and may be more efficiently implemented. The price to be paid is that communication between arbitrary membranes is not straightforward and has to be implemented with small steps of passing objects from region to region to reach the final destination. In K one has a somehow mixed setting: K uses membranes to enforce local rewriting, but the matching rules are global.

Three main classes of P-systems have been extensively studied:

P-systems as transition systems ["classical" P-systems]
P-systems as communicating systems [symport/antiport P-systems]
P-systems as structure-evolving systems [P-systems with active membranes]

We present formalizations in K of these three basic types of P-systems and of many key elements used in the plethora of P-systems found in the literature. We believe that this is enough evidence that K can serve as a suitable semantic framework for defining P-systems in particular, and, in general, for experimenting with new parallel programming languages based on paradigms from natural science, like P-systems, for which efficient implementations are notoriously difficult to develop. We make two additional contributions:

- We extend P-systems from a setting with unstructured objects (or using a simple monoid structure on objects, i.e., strings) to one where objects are given by arbitrary equational specifications. Most data types may be represented by algebraic specification techniques, hence one can use this line to incorporate complex data types into P-systems.
- We have developed a running environment for P-systems using the embedding techniques discussed in this paper and the implementation of K in Maude. The paper includes a few experiments with both unstructured and structured objects which demonstrate the large increase in expressivity and performance due to adding structure to objects.

The paper is organized as follows. Sections 2 and 3 briefly introduce K and P-systems, respectively, referring the reader to the literature for more detail. Sections 4, 5 and 6 show how the three types of P-systems above are defined in K. Section 7 discusses our implementation and Section 8 concludes the paper.

2 K

K [23] is a framework for defining programming languages based on rewriting logic RWL [15], in the sense that it has two types of sentences: *equations* for structural identities and *rules* for computational steps. It has an implementation in Maude and it allows

Int ::=	... all integer numbers		
Bool ::=	true | false		
Name ::=	all identifiers; to be used as names of variables		
Val ::=	*Int*		
AExp ::=	*Val* | *Name*		
|	*AExp* + *AExp*	[*strict*, *extends* $+_{Int \times Int \to Int}$]	(r_1)
BExp ::=	*Bool*		
|	*AExp* $\leq$ *AExp*	[*seqstrict*, *extends* $\leq_{Int \times Int \to Bool}$]	(r_2)
|	not *BExp*	[*strict*, *extends* $\neg_{Bool \to Bool}$]	(r_3)
|	*BExp* and *BExp*	[*strict*(1)]	(r_4)
Stmt ::=	*Stmt*; *Stmt*	$[s_1; s_2 = s_1 \curvearrowright s_2]$	(r_5)
|	*Name* := *AExp*	[*strict*(2)]	(r_6)
|	if *BExp* then *Stmt* else *Stmt*	[*strict*(1)]	(r_7)
|	while *BExp* do *Stmt*		(r_8)
|	halt *AExp*	[*strict*]	(r_9)
Pgm ::=	*Stmt*; *AExp*		

Table 2.1 K-annotated syntax of IMP.

for a fast development of efficient interpretors. We briefly describe the basic concepts and notations used in K (see [23] and the referenced website for a detailed presentation of K; ask the author for the current version of a draft book) using as an example a simple concurrent language. We start with the simple imperative language IMP defined in Tables 2.1 and 2.2. Then, we extend IMP with threads and call the resulting language CIMP.

Annotating syntax. The plus operation (r_1) is said to be "strict" in both arguments/subexpressions (i.e., they may be evaluated in any order), while the conditional (r_7) is strict only in its first argument (i.e., the boolean expression has to be evaluated first); finally, in "less-than" (r_2) the evaluation is "seqstrict" (i.e., the arguments are evaluated in the sequential order). Some operations "extend" other operations on primitive data types (which may be computed with external libraries, for example). All attributes can be desugared with equations or rules (if they are not already equations or rules); they are nothing but notational convenience. For example, the extends attribute in (r_1) desugars to rule "$i_1 + i_2 \to i_1 +_{Int \times Int \to Int} i_2$". The desugaring of strictness is explained shortly.

Semantics. The K semantics is defined with equations and rules that apply on a nested *cell* (or *membrane*) structure, each cell containing either multisets or lists of elements. A K semantics of IMP is described in Table 2.2. K *configurations* are specified using cells $\langle\!| \; S \; |\!\rangle_h$, where h is a cell index and S is any term, in particular a multiset or a list of possibly other cells. Set[S] and List[S] denote multisets and lists of terms of type

S; by default, the empty set or list is denoted by a dot "$\cdot$", and multiset elements are separated by space while list elements are separated by comma. If one wants a different separator or unit, or wants to emphasize a default one, then one can specify it as sub- and/or super-script; for example, $\mathsf{List}^{\cdot}_{\curvearrowright}[S]$ denotes "$\curvearrowright$"-separated lists of elements of type S of unit "$\cdot$". The syntactic category K stays for *computations* and typically has a "$\curvearrowright$"-separated list structure of *computational tasks*, with the intuition that these are processed sequentially. What "processed" means depends upon the particular definition. Strictness attributes are syntactic sugar for special equations allowing subcomputations to be "scheduled for processing". The strictness attributes in Table 2.1 correspond to ($k, k_1, k_2 \in K$, $r_1 \in$ *KResult*, $x \in$ *Name*):

$k_1 + k_2 = k_1 \curvearrowright \square + k_2$ $\quad$ k_1 and $k_2 = k_1 \curvearrowright \square$ and k_2

$k_1 + k_2 = k_2 \curvearrowright k_1 + \square$ $\quad$ $x := k = k \curvearrowright x := \square$

$k_1 \leq k_2 = k_1 \curvearrowright \square \leq k_2$ $\quad$ if k then k_1 else $k_2 = k \curvearrowright$ if $\square$ then k_1 else k_2

$r_1 \leq k_2 = k_2 \curvearrowright r_1 + \square$ $\quad$ halt $k = k \curvearrowright$ halt $\square$

not $k = k \curvearrowright$ not $\square$

The square "$\square$" is part of auxiliary operator names called *computation freezers*; for example, "$\square + \_$" freezes the computation k_2 in the first equation above.

In K, the following derived notations are used to indicate an occurrence of an element in a cell: $\langle\!|\ S\ |\!\rangle_h$ - at the top; $\langle\!|\ S\ |\!\rangle_h$ - at the very end; $\langle\!|\ S\ |\!\rangle_h$ - anywhere. The first two notations are useful when the configuration is a list, but they are not used in the representation of P-systems in K described in this paper. Rules can also be written in *contextual form* in K, where subterms to be replaced are underlined and the terms they are replaced by are written underneath the line. For example, the assignment rule

$$\langle\!|\underline{x := v}|\!\rangle_k \ \langle\!|(x, \underline{\_})|\!\rangle_{\mathit{state}} \qquad \begin{matrix} \cdot \\ v \end{matrix}$$

reads as follows (underscore "_" matches anything): if an assignment "$x := v$" is at the top of the computation, then replace the current value of x in the state by v and dissolve

KResult ::= *Val*	$\langle\!\|\underline{x := v}\|\!\rangle_k \ \langle\!\|(x, \underline{\_})\|\!\rangle_{\mathit{state}}$ $\quad \begin{matrix} \cdot \\ v \end{matrix}$
K ::= *KResult* $\mid \mathsf{List}^{\cdot}_{\curvearrowright}[K]$	if true then s_1 else $s_2 \rightarrow s_1$
Config ::= $\langle\!\|K\|\!\rangle_k \mid \langle\!\|\mathsf{Set}[(\mathit{Name}, \mathit{Val})]\|\!\rangle_{\mathit{state}}$	if false then s_1 else $s_2 \rightarrow s_2$
$\mid \ \langle\!\|\mathsf{Set}[\mathit{Config}]\|\!\rangle_\top$	$\langle\!\|$ while b do s $\|\!\rangle_k$
$\langle\!\|\underline{x}\|\!\rangle_k \ \langle\!\|(x, v)\|\!\rangle_{\mathit{state}}$ $\quad v$	if b then $(s;$ while b do $s)$ else $\cdot$
true and $b \rightarrow b$, $\quad$ false and $b \rightarrow$ false	$\langle\!\|$halt $i\|\!\rangle_k \rightarrow \langle\!\|i\|\!\rangle_k$

Table 2.2 K configuration and semantics of IMP.

the assignment statement. We prefer to use the more conventional notation "$l \rightarrow r$" instead of "$\frac{l}{r}$" when the entire term changes in a rule.

Extending IMP with threads. We extend IMP with threads and call the resulting language CIMP (from concurrent IMP). We only add a spawning statement to the syntax of the language, without any explicit mechanisms for synchronization.

Stmt ::= ... | spawn *Stmt*

Interestingly, all we have to do is to add a K rule for thread creation and nothing from the existing definition of the K semantics of IMP has to be changed. Here is the rule for spawning:

$$\langle\!| \frac{\mathsf{spawn}(s)}{\cdot} |\!\rangle_k \quad \frac{\cdot}{\langle\!| s |\!\rangle_k}$$

Therefore, multiple $\langle\!| \_ |\!\rangle_k$ cells can live at the same time in the top cell, one per thread. Thanks to the concurrent rewriting semantics of K, different threads can access (read or write) different variables in the state at the same time. Moreover, two or more threads can concurrently read the same variable. This is precisely the intended semantics of multithreading, which is, unfortunately, not captured by SOS definitions of CIMP, which enforce an interleaving semantics based on nondeterministic linearizations of the concurrent actions. While K allows a concurrent semantics to a language, note that it does *not* enforce any particular "amount of concurrency"; in particular, K's concurrent rewriting relation, denoted "$\Rightarrow$", includes any number of rewrites that can be executed concurrently, from one rewrite to a maximal set of rewrites; thus, an interleaved execution or a maximally parallel one are both valid rewrite sequences in K.

Once a thread is terminated, its empty cell (recall that statements are processed into empty computations) will never be involved into any matching, so it plays no role in the future of the program, except, perhaps, that it can overflow the memory in actual implementations of the K definition. It is therefore natural to cleanup the useless cells with an equation of the form:

$$\langle\!| \cdot |\!\rangle_k = \cdot$$

Synchronization mechanisms through lock acquire and release, as well as through rendezvous barriers, are discussed in detail in [23].

3 Membrane systems

Membrane systems (or P-systems) are computing devices abstracted from the structure and the functioning of the living cell [1].

In classical *transition P-systems* [20], the main ingredients of such a system are the

membrane structure (a hierarchical cell-like arrangement of membranes[15]), in the compartments of which *multisets* of symbol-objects evolve according to given *evolution rules*. The rules are localized, associated with the membranes (hence, with the compartments), and they are used in a *nondeterministic maximally parallel* manner (a unique clock is assumed, the same for all compartments). A computation consists of a sequence of transitions between system configurations leading to a halting configuration, where no rule can be applied. With a halting computation one associates a result, usually in the form of the number of objects present in a distinguished membrane. Thus, such a system works with numbers inside (multiplicities of objects in compartments) and provides a number as the result of a computation.

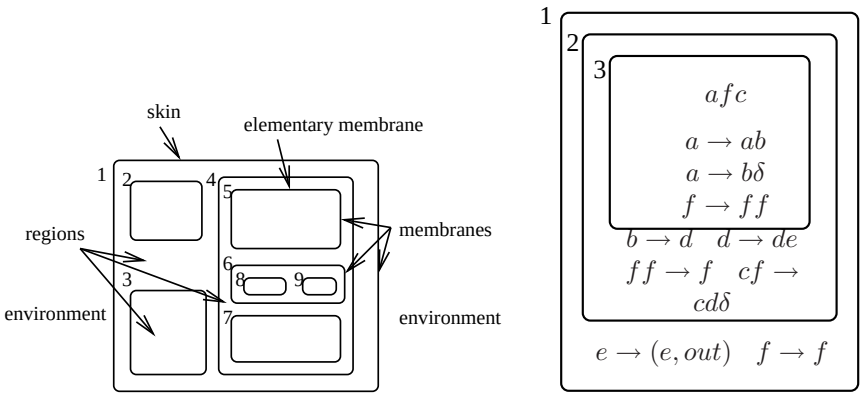


Fig. 3.1 A membrane system (a) and a "classical" P system (b).

From a different perspective, P-systems may be seen as communicating systems. In this view, a P-system, better known as *symport/antiport P-system* [19], computes by moving objects through membranes, in a way inspired by biology. The rules are of the forms (x, in) and (x, out) (*symport* rules, with the meaning that the objects specified by x enter, respectively exit, the membrane with which the rule is associated), and $(x, out; y, in)$ (*antiport* rules: the objects specified by x exit and those specified by y enter the membrane at the same time). By rules of these types associated with the skin membrane of the system, objects from the environment can enter the system and, conversely, objects from the system can be sent out into the environment. One can also use *promoters* (*inhibitors*) associated with the symport/antiport rules, in the form of objects which must be present in (resp. absent from) a compartment in order to allow the associated rule to be used.

Finally, a feature which may be added to any of the previous types of P-systems is the possibility to dynamically change the membrane structure. The resulting P-systems are called *P-systems with active membranes* [21].

[15] See [8], for a similar structure.

4 P-systems as transition systems (classical P-systems)

This is the classical type of P-systems, originally introduced in [20]. In this model, each membrane has an associated set of rules. The objects can travel through membranes and they may be transformed by the rules (the rules can create or destroy objects). A membrane may be dissolved and its objects flood into the parent region, while the rules vanish.

4.1 Basic transition P-systems A *transition P-system*, of degree $m \geq 1$, is formally defined by a tuple

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

where: (1) O is an alphabet of *objects*; (2) $C \subseteq O$ is the set of catalysts; (3) μ is a membrane structure (with the membranes bijectively labeled by natural numbers $1, \ldots, m$); (4) $w_1, \ldots, w_m$ are multisets over O associated with the regions $1, \ldots, m$ of μ, represented by strings from O^* unique up to permutations; (5) $R_1, \ldots, R_m$ are finite sets of rules associated with the membranes $1, \ldots, m$; the rules are of the form

$u \rightarrow \overline{v}$ or $u \rightarrow \overline{v}\delta$
with $u \in O^+$ and $\overline{v} \in (O \times Tar)^*$, where $Tar = \{here, in, out\}$

(6) i_o is the label of the output membrane, an elementary one in μ or $i_o = 0$, indicating that the collecting region is the environment. When δ is present in the rule, its application leads to the dissolution of the membrane and to the abolishment of the rules associated with the membrane just dissolved.

A membrane is denoted by $[_h\ ]_h$. By convention, $[_h u]_h$ denotes a membrane with u present in the solution (among other objects). Starting from the *initial configuration*, which consists of μ and $w_1, \ldots, w_m$, the system passes from one configuration to another by applying a transition, i.e., the rules from each set R_i in a *non-deterministic and maximally parallel* way. A sequence of transitions is called a *computation*; a computation is *successful* if and only if it halts. With a successful computation one associates a *result*, in the form of the number of objects present in membrane i_o in the halting configuration.

An example is presented in Fig. 3.1(b) – it computes/generates the square numbers $(k+1)^2, k \geq 0$. When a rule with δ is applied, the corresponding membrane and its rules are dissolved and its current objects are flooded into the parent region. A typical, terminating evolution of this system is as follows: It starts in membrane 3 (the other membranes have no objects), then membrane 3 is dissolved and the evolution continues in membrane 2, and when this is dissolved, the final stage of the execution is in membrane 1 (the skin membrane). In membrane 3, the first two rules have a conflict on a: when the 2nd rule is applied, object a, as well as membrane 3, disappear; the rule for f

is independent and has to be used each time another rule is applied; to conclude, when membrane 3 disappears, we are left with b^{k+1} and $f^{2^{k+1}}$ objects which are flooded into membrane 2. In membrane 2, at each cycle the f's are divided by 2 and, in parallel, a copy of each b (now rewritten in d) is created. Finally one gets $(k+1)^2$ copies of object e when membrane 2 is dissolved, which are passed to membrane 1 and then into the external environment.

4.2 Basic transition P-systems in K Given a P-system Π, we define a K-system $K(\Pi)$, as follows:

- A membrane $[_h S]_h$, where S is its contents, is represented as[16]

  $$\langle\!| \; S \; |\!\rangle_h$$

  The top configuration is $\langle\!| \; \langle\!| \; |\!\rangle_{skin} \quad \langle\!| \; |\!\rangle_{env} \; |\!\rangle_\top$, including a representation of the objects in the environment.
- The rules in Π are represented as global rules in $K(\Pi)$, their localization being a side-effect of membranes name matching. The evolution rules are reduced to two basic rules used in a membrane $[_h \,]_h$ and are represented in K as follows:
  - $u \to \overline{v}$, with $u \in O^+, \overline{v} \in (O \times Tar)^*$, where $Tar = \{here, in, out\}$
    For a $\overline{v} \in (O \times Tar)^*$, let v be its restriction to O. Next (recall that composition is commutative), let v be written as $v_h v_i v_o$, where v_h contains the objects that remain in the membrane $[_h \,]_h$, v_i those that enter into internal membranes, and v_o those that leave out the membrane. Then, the associated rule in K is the following: for any $k \geq 1$ and $v_i = v_1 \ldots v_k$ (with nonempty v_j's) we have a rule

    $$\langle\!| \quad \underline{u}_{v_h} \quad \langle\!| \; \underline{\cdot}_{v_1} \; |\!\rangle \ldots \langle\!| \; \underline{\cdot}_{v_k} \; |\!\rangle \quad |\!\rangle_h \quad \underline{\cdot}_{v_o}$$

  - $u \to u\delta$ (δ indicates that the membrane is dissolved)

    $$\langle\!| \; u \; z \; |\!\rangle_h \to u \; z$$

For the first rule $u \to \overline{v}$, a second possibility is to perform two steps: move v_h, v_o first, then for all remaining tagged objects (v_r, in) use a matching rule $\langle\!| \; (v_r, in) \; \langle\!| \; \underline{\cdot}_{v_r} \; |\!\rangle \; |\!\rangle$.

The rules for object movement and membrane dissolution may be combined. For instance, the rule $u \to \overline{v}\delta$, with $v = v_h v_i v_o$, as above but for $k = 1$,[17] may be represented as

[16] Another representation may be $\langle\!| \; \langle\!| \; h \; |\!\rangle_{id} \; S \; |\!\rangle_{cell}$, if one prefers a single cell type. This can also be an implementation optimization, to avoid structured cell labels.

[17] For simplicity, we describe the case $k = 1$ where all objects which enter an internal membrane enter into the same membrane i.

$$(\!|\ u\ (\!|w|\!)_i\ z\ |\!)_h \rightarrow v_h\ (\!|w\ v_i|\!)_i\ z\ v_o$$

In this interpretation, first the objects are moved out from the membrane, then the membrane is dissolved. One can go the other way round, first to dissolve the membrane and then to move the objects out; the K rule is

$$\langle\ \frac{(\!|\ u\ (\!|\ w\ |\!)_i\ z\ |\!)_h}{v_h\ (\!|\ wv_i\ |\!)_i\ z}\ \rangle\_\ \frac{\cdot}{v_o}$$

Parallel rewriting in K. The rewriting logic in K extends the one in RWL by allowing for overlapping rules which overlap on parts that are not changed by these rule (on "read-only" parts). Such an extension is needed, e.g., when two threads read the store at the same time.

As an extension of the rewriting mechanism in RWL, the rewriting in K allows for the application of an arbitrary number of rewriting rules in a step. However, it does not constrain the user to use a "maximal parallel" rewriting like in the case of P-systems. Such an option may be handled at the meta-level by selecting an appropriate strategy for applying the rules. Actually, there is little evidence that a "maximal parallel" strategy is present in the living cells - it is more like a hypothesis to keep the evolution simpler and to find results in the theoretical model. We see here another incarnation of the classical dichotomy between synchronous and asynchronous systems: the latter are more suited for real applications, while the former are easier to understand.

4.3 Variations of transition P-systems

Object movement. We present a few variations of the object movement rules found in the literature (and included in the survey [22]), then we describe their associated rules in K.

- (deterministic in) In this variant, one uses in_j instead of a simple in to indicate that an object goes into the internal membrane j. Its K-translation is

$$\langle\ \frac{u}{v_h}\ \ \langle\ \frac{\cdot}{v_1}\ \rangle_{j_1} \ldots \langle\ \frac{\cdot}{v_k}\ \rangle_{j_k}\ \rangle_h\ \ \frac{\cdot}{v_o}$$

  where v_h, v_i, v_o are as above and $v_1, \ldots, v_k$ are the objects in v that go into the internal membranes $j_1, \ldots, j_k$, respectively.

- (polarities) One can use classes of membranes with polarities, say $+/0/-$. The newly created objects may have $+/0/-$ polarities as well and those with $+/-$ polarities go into internal membranes with opposite polarities, while those with 0 polarity may stay or goes outside. This is a case in between the above two extreme alternatives: complete freedom to go in any internal membrane and precise target

for each object. It is, therefore, easy to provide in K definitions for the two cases above. There are various ways to add algebraic structure for polarities. For example, one way pair each datum and each membrane label with a polarity; in the case of data we write the polarity as a superscript, e.g., a^+ or a^-, while for the membranes we write it as a superscript of the membrane, e.g., $\langle\!\langle\ u\ \rangle\!\rangle_h^+$ or $\langle\!\langle\ u\ \rangle\!\rangle_h^-$. The membrane polarity may be changed, as well. For example, to describe that in the presence of objects u the polarity is changed from + to $-$ one can use the K rule

$$\langle\!\langle\ u\ \rangle\!\rangle_h^+ \rightarrow \langle\!\langle\ u\ \rangle\!\rangle_h^- \quad \text{or, equivalently,} \quad \langle\!\langle\ u\ \rangle\!\rangle_h^{\pm}$$

- (arbitrary jumps) In this variant, one can directly move an object into any other membrane. The rule, written as $[_h u]_h \rightarrow [_{h'} v]_{h'}$. To represent this rule in K we use explicit rules for matching the membranes at different levels (the notation $\langle\!\langle^*\ \ ^*\rangle\!\rangle$ means a match in any recurrently included cell, not only in the current one):
  - $*$ if the membranes are not contained one into the other, then
  $$\langle\!\langle\ \langle\!\langle^*\ \underset{\cdot}{\underline{u}}\ ^*\rangle\!\rangle_h \quad \langle\!\langle^*\ \underset{v}{\underline{\cdot}}\ ^*\rangle\!\rangle_{h'}\ \rangle\!\rangle$$
  - $*$ if the jump is into an enclosed membrane, then
  $$\langle\!\langle\ \underset{\cdot}{\underline{u}} \quad \langle\!\langle^*\ \underset{v}{\underline{\cdot}}\ ^*\rangle\!\rangle_{h'}\ \rangle\!\rangle_h$$
  - $*$ if the jump is into an outside membrane, then
  $$\langle\!\langle\ \underset{v}{\underline{\cdot}} \quad \langle\!\langle^*\ \underset{\cdot}{\underline{u}}\ ^*\rangle\!\rangle_h\ \rangle\!\rangle_{h'}$$

  Instances of this rule capture the particular cases of in^*/out^*, notation used in P-systems to indicate a movement into an elementary/skin membrane.

Membrane permeability. In the standard setting, a membrane is passive (label 1) and the specified objects can pass through it. The membrane can be dissolved (label 0), as well. One can add impenetrable membranes (label 2), as well. The status of the membranes may be dynamically changed as a side-effect of applying reaction rules.

This case is similar to the case of polarities: One can use pairs (h, i) to represent the membrane id and its permeability level. The rules are simple variations of the basic P-systems evolution rules and may be easily handled in K.

Catalysts. The role of catalysts may be viewed in two opposite ways: (1) either they may be seen as a sine-qua-non ingredient for a reaction $a \rightarrow b$ to take place; or (2) they may be seen as a way to control the parallelism, by restricting a free reaction $a \rightarrow b$ to the number of occurrences of the catalyst. In a further extension, catalysts may move from a membrane to another, or may change their state (each catalyst is supposed to have a finite number of isotopic forms).

Catalysts are just objects, hence the translation in K is straightforward.

Rules with priorities. Capturing various control mechanisms on applying the rules is a matter of strategies. Strategies are commonly held separate of rules in many rewriting approaches. One important way to restrict the maximal parallelism convention is to apply the rules according to their priorities. In a strong version, only the rule with the highest priority applies; in a weak version, when all possible applications of the highest priority rule have been resolved, a next rule with less priority is chosen, and so on.

In the current implementation of P-systems over K and Maude, priorities are handled at Maude "meta-level" and using the corresponding priority algorithm to capture a P-system maximal parallel step. (K has not developed particular strategies and uses the strategies inherited from Maude.)

Promoters/inhibitors. The presence of promoters/inhibitors may be seen as additional context for rules to apply. In the case of promoters, a reaction rule $l \rightarrow r$ in membrane h applies only if the objects in a promoter set z are present in the solution (they should be different from those in l):

$$\langle\!\langle \underset{r}{\underline{l}}\ z \rangle\!\rangle_h$$

The case of inhibitors is opposite: in the presence of the objects in z the rule cannot apply. The case of inhibitors requires a K rule with a side condition

$$\langle\!\langle \underset{r}{\underline{l}}\ z \rangle\!\rangle \quad \text{where } z \not\subseteq x$$

Complex side condition like the above are handled by means of conventional conditional rewrite rules in our Maude implementation of K.

Border rules. Border rules are particular object evolution rules of the following type $xu\lfloor_i vy \rightarrow xu'\lfloor_i v'y$. They allow to test and modify the objects from two neighboring regions. Such a rule may be represented in K by

$$x\ \underset{u'}{\underline{u}}\ \langle\!\langle\ \underset{v'}{\underline{v}}\ y \rangle\!\rangle_{\_}$$

5 P-systems as communicating systems (symport/antiport P-systems)

This type of P-systems was introduced in [19]. In this variant, the environment is considered to be an inexhaustible source of objects of any type. The evolution rules are called symport/antiport rules and only move the objects through the membranes (they do not create or destroy objects).

Basic symport/antiport P-systems. A *symport/antiport P system*, of degree $m \geq 1$, is formally defined by a tuple

$$\Pi = (V, T, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m, i_o),$$

where: (1) V is an alphabet of *objects*; (2) $T \subseteq V$ is the terminal alphabet; (3) μ is a membrane structure (with the membranes bijectively labeled by natural numbers $1, \ldots, m$); (4) $w_1, \ldots, w_m$ are multisets over V associated with the regions $1, \ldots, m$ of μ, represented by strings from V^*; (5) $E \subseteq V$ is the set of objects which are supposed to appear in an arbitrarily large number of copies in the environment; (6) $R_1, \ldots, R_m$ are finite sets of symport and antiport rules associated with the membranes $1, \ldots, m$:

- a symport rule is of the form (x, in) or (x, out), where $x \in V^+$, with the meaning that the objects specified by x enter, respectively exit, the membrane, and
- an antiport rule is of the form $(x, in; y, out)$, where $x, y \in V^+$, which means that x is taken into the membrane region from the surrounding region and the multiset y is sent out of the membrane;

(7) i_o is the label of the output membrane, an elementary one in μ.

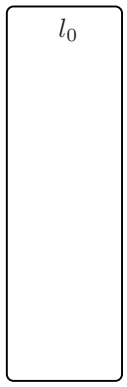
With the symport/antiport rules one can associate *promoters* $(x, in)|_z$, $(x, out)|_z$, $(x, out; y, in)|_z$, or *inhibitors* $(x, in)|_{\neg z}$, $(x, out)|_{\neg z}$, $(x, out; y, in)|_{\neg z}$, where z is a multiset of objects – such a rule is applied only if z is present, respectively not present.

Starting from the *initial configuration*, which consists of μ and $w_1, \ldots, w_m, E$, the system passes from one configuration to another by applying the rules from each set R_i in a *non-deterministic and maximally parallel* way.[18] A sequence of transitions is called a *computation*; a computation is *successful* if and only if it halts. With a successful computation we associate a *result*, in the form of the number of objects from T present in membrane i_o in the halting configuration.

Example. An example is presented in Fig. 5.2. It describes how symport/antiport P-systems may simulate counter machines – it is well known that counter machines with at least 2 counters are computationally universal, see [11]. A counter machine uses a finite number of counters and a program consisting of labeled statements. Except for the begin and halt statements, the other statements perform the following actions: (1) increase a counter by one, (2) decrease a counter by one, or (3) test if a counter is zero. For the simulation in Fig. 5.2, one uses an equivalent definition of counter machines where the statements are of one of the following two types:

(a) $l_1 : (add(r), l_2, l_3)$ (add 1 to r and nondeterministically go to l_2 or l_3)
(b) $l_1 : (sub(r), l_2, l_3)$ (if r is not 0, subtract 1 and go to l_2, else go to l_3)

[18] We recall that the environment is supposed inexhaustible, at each moment all objects from E are available in any number of copies we need.



$E = \{a_r | 1 \leq r \leq m\} \cup \{l, l', l'', l''', l^{iv} | l \in B\}$
for $l_1 : (\texttt{add}(r), l_2, l_3)$:
- $(l_1, out; a_r l_2, in)$
- $(l_1, out; a_r l_3, in)$
for $l_1 : (\texttt{sub}(r), l_2, l_3)$:
- $(l_1, out; l_1' l_1'', in)$
- $(l_1' a_r, out; l_1''', in)$
- $(l_1'', out; l_1^{iv}, in)$
- $(l_1^{iv} l_1''', out; l_2, in)$
- $(l_1^{iv} l_1', out; l_3, in)$
for l_h : `halt`:
- (l_h, out)

Fig. 5.2 A "symport/antiport" P system.

The simulation works as follows: The statement to be next executed is in the (unique) cell, while the others stay outside. At each step, the current statement leave the cell and the one to be next executed goes inside the cell. During this process, the counter associated to the statement goes updated. The processing of a type (b) statement is slightly more complicate as a trick is to be used to check if the counter is zero, see [22].

Basic symport/antiport P-systems in K. The previous representation in K of transition P-systems and their variations almost completely covers this new type of P-systems. What is not covered is the behavior of the environment. The K rule is actually an equation

$$\langle\!\langle\ x\ \rangle\!\rangle_{env} = \langle\!\langle\ x\ \rangle\!\rangle_{env}\ x$$

Variations of symport/antiport P-systems. Most of the variations used for transition P-systems can be used here, as well.

6 P-systems with active membranes

The third main class of P-systems brings an important additional feature: the possibility to dynamically change the membrane structure. The membranes can evolve themselves, either changing their characteristics or getting divided.

6.4 Basic P-systems with active membranes A *P-system with active membranes* [21] is formally defined by a tuple

$$\Pi = (O, H, \mu, w_1, ..., w_m, R)$$

where: (1) $m \geq 1$ (the initial degree of the system); (2) O is the alphabet of objects; (3) H is a finite set of labels for membranes; (4) μ is a membrane structure, consisting of m

membranes having initially neutral polarizations, labeled (not bijectively) with elements of H; (5) $w_1, \ldots, w_m$ are strings over O, describing the multisets of objects placed in the m regions of μ; (6) R is a finite set of developmental rules, of the following forms:

- (a) object evolution rules: for $h \in H, e \in \{+, -, 0\}, a \in O, v \in O^*$,

$$[_h a \rightarrow v]^e_h$$

- (b) "in" communication rules: for $h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O$,

$$a[_h\ ]^{e_1}_h \rightarrow [_h b]^{e_2}_h$$

- (c) "out" communication rules: for $h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O$,

$$[_h a]^{e_1}_h \rightarrow [_h\ ]^{e_2}_h b$$

- (d) dissolving rules: for $h \in H, e \in \{+, -, 0\}, a, b \in O$,

$$[_h a]^e_h \rightarrow b$$

- (e) division rules (elementary membranes, only): for $h \in H, e1, e2, e3 \in \{+, -, 0\}$, $a, b, c \in O$

$$[_h a]^{e_1}_h \rightarrow [_h b]^{e_2}_h [_h c]^{e_3}_h$$

The objects evolve in the maximally parallel manner, while each membrane can be involved in only one rule of types (b)-(e). More precisely, first the rules of type (a) are used, and then the other rules.

The label set H has been specified because it is allowed to change the membrane labels. Notice that one uses a dictionary of rules, each label in H coming with its own set of rules. For instance, a division rule can be of the more general form

- (e') general division: for $h_1, h_2, h_3 \in H, e_1, e_2, e_3 \in \{+, -, 0\}, a, b, c \in O$

$$[_{h_1} a]^{e_1}_{h_1} \rightarrow [_{h_2} b]^{e_2}_{h_2} [_{h_3} c]^{e_3}_{h_3}$$

One can consider variations as the possibility of dividing membranes in more than two copies, or even of dividing non-elementary membranes. Therefore, in P-systems with active membranes the membrane structure evolves during the computation not only by decreasing the number of membranes by dissolution, but also increasing it by division.

6.5 Basic P-systems with active membranes in K Except for membrane division, P-systems with active membranes are similar to transition P-systems, hence we can borrow the previous translation in K. However, for the sake of clarity, we prefer to give the K-representation for the full set of rules (a)-(e). A membrane with polarity is denoted by pairs (h, e) with $h \in H$ and $e \in \{+, -, 0\}$.

- object evolution rules: $[_h a \to v]_h^e$ ($h \in H, e \in \{+,-,0\}, a \in O, v \in O^*$)

$$\langle\!\langle \begin{array}{c} a \\ \hline v \end{array} \rangle\!\rangle_h^e$$

- "in" communication rules: $a[_h]_h^{e_1} \to [_h b]_h^{e_2}$ ($h \in H, e_1, e_2 \in \{+,-,0\}, a, b \in O$)

$$\begin{array}{c} a \\ \hline \cdot \end{array} \quad \langle\!\langle \begin{array}{c} \cdot \\ \hline b \end{array} \rangle\!\rangle_h^{\frac{e_1}{e_2}}$$

- "out" communication rules: $[_h a]_h^{e_1} \to [_h]_h^{e_2} b$ ($h \in H, e_1, e_2 \in \{+,-,0\}, a, b \in O$)

$$\langle\!\langle \begin{array}{c} a \\ \hline \cdot \end{array} \rangle\!\rangle_h^{\frac{e_1}{e_2}} \quad \begin{array}{c} \cdot \\ \hline b \end{array}$$

- dissolving rules: $[_h a]_h^e \to b$ ($h \in H, e \in \{+,-,0\}, a, b \in O$)

$$(\!| \; a \quad x \; |\!)_h^e \to b \quad x$$

- division rules for elementary membranes: $[_h a]_h^{e_1} \to [_h b]_h^{e_2} [_h c]_h^{e_3}$ ($h \in H, e1, e2, e3 \in \{+,-,0\}, a, b, c \in O$)

$$(\!| \; a \; |\!)_h^{e_1} \to (\!| \; b \; |\!)_h^{e_2} \quad (\!| \; c \; |\!)_h^{e_3}$$

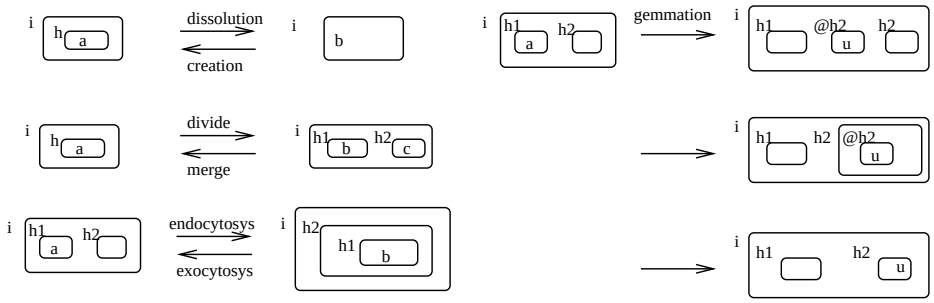


Fig. 6.3 Membrane handling operations.

6.6 Variations of P-systems with active membranes

Membrane creation. This rule is $a \to [_h v]_h$, i.e., after its application a new membrane is inserted into the system. The rules in the new membrane depend on h and they are taken from a dictionary. The K rule is

$$a \to (\!| \; v \; |\!)_h$$

Merging of membranes. This rule is $[_h x]_h [_{h'} y]_{h'} \rightarrow [_{h''} z]_{h''}$, allowing to merge the contents of two neighboring membranes (the rules for $[_{h''} ..]_{h''}$ are taken from a dictionary). The K rule is

$$\langle\!| \; x \; |\!\rangle_h \quad \langle\!| \; y \; |\!\rangle_{h'} \rightarrow \langle\!| \; z \; |\!\rangle_{h''}$$

Split of membranes. This operation is opposite to merge. Its format is $[_{h''} z]_{h''} \rightarrow [_h x]_h [_{h'} y]_{h'}$ and the K rule is

$$\langle\!| \; z \; |\!\rangle_{h''} \rightarrow \langle\!| \; x \; |\!\rangle_h \quad \langle\!| \; y \; |\!\rangle_{h'}$$

One appealing version is to put into a membrane the objects of a given type and in the other the remaining ones.

Endocytosys and exocytoses. Endocytosys is a rule $[_h x]_h [_{h'}]_{h'} \rightarrow [_{h'} [_h y]_h]_{h'}$, i.e., in one step a membrane and its contents enter into a neighboring membrane. The K rule is

$$\frac{\langle\!| \; x \; |\!\rangle_h}{\cdot} \quad \langle\!| \; \frac{\cdot}{\langle\!| \; y \; |\!\rangle_h} \; |\!\rangle_{h'}$$

Gemmation. One can encapsulate into a new membrane $[_{@h} \,]_{@h}$ a part u of the solution to be carried to membrane $[_h \,]_h$. The new membrane $[_{@h} u]_{@h}$ travels through the system, being safe for its contents. By convention, it travels with the speed of one membrane per clock cycle following the shortest path towards the destination.

The final result may be described as in the case of general object jumps. What is different here is the step-by-step journey of $[_{@h} u]_{@h}$. This can be done using the shortest path from h' to h in the tree associated to this membrane system.[19] The details are left to the reader.

7 Implementation, experiments

The intractability of P-systems with plain objects. While P-systems is a model with massive parallelism, its huge potential is not fully exploited by current approaches due to the lack of object structure.[20] The illustrating example of computing n^2 with the P-system in Fig. 3.1(b) is not a fortunate one. For instance, to represent 999 one needs 999 objects in the membrane and the computation ends up with 998001 objects in the final region; however, during computation an exponential mechanism to record the number of steps is used and in an intermediary soup there are more than 2^{999} objects, significantly

[19] One does not consider the complicated case when the delivery gets lost into the system due to a reconfiguration of the membrane structure.

[20] A few P-systems with particular structured objects (strings, conformons) are presented in [20, 12].

more than the atoms in the Universe.[21] There are other sophisticated representation mechanisms in cells which may be used to achieve fast and reliable computations of interest for cells themselves. As shown below, with structured objects we break ground and achieve fast computations for P-systems with structured objects.

A K/Maude implementation. We have developed an application for running P-systems using our embedding of P-systems in K and the implementation of K in Maude (Maude [10] can be downloaded at http://maude.cs.uiuc.edu/). The application can be accessed online at

`http://fsl.cs.uiuc.edu/index.php/Special:MaudeStepperOnline`

One can choose the examples in the p-system directory. The application allows to run a P-system blindly, or in an interactive way; another option is to ask for displaying the transition graph. (The options for interactive running or graph display make sense for small examples, only.)

Results: structured vs. unstructured objects. We have performed a few experiments, both with plain, unstructured objects and with structured ones.

For the former case, we implemented the P-system in Fig. 3.1(b). (We have already commented on its intractability above.) We run experiments on our server with the following constraints: up to 2 minutes and using no more than 500 MB of RAM. With these constraints, we were able to compute n^2, but *only up to* $n = 18$. The results are presented in Table 7.3(a).

For the latter case (structured objects) we limited ourselves to natural numbers and run two experiments with P-systems for computing factorial function and for looking for prime numbers using Eratosthenes sieve. In both cases, we were able to run large experiments: in the first case, we were able to compute $3000!$ (a number with 12149 digits) within the given constraints; in the second case, all prime numbers less than 1500 where found in no more than 1 minute. The results are collected in Table 7.3(b),(c). The transition graph for the example with prime numbers up to $n = 10$ is displayed in Fig. 7.4.

The P-systems for the last two examples are flat, the computation being similar to that used in Γ-programs. It is possible to describe P-systems with more membranes for this problem and to find the resulting speedup, but this is out of the scope of the current paper.

[21] The Universe is estimated to 4×10^{79} hydrogen atoms, while 2^{999} is roughly 10^{300}.

square	time(ms)	rewritings	parallel steps
6	11	2316	13
10	123	20012	21
15	3882	562055	31
18	32294	4463244	37
19	failure		

(a)

factorial	time(ms)	no rew.	parallel steps
10	0	93	4
100	8	744	7
1000	545	7065	10
3000	6308	21079	12
3500	failure		

(b)

primes	time(ms)	no rew.	parallel steps
10	1	156	2
100	33	16007	6
1000	19007	2250684	14
1500	50208	5402600	15
2000	failure		

(c)

Table 7.3 Runs for P-systems with and without data structure on objects.

8 Related and future work, conclusions

A similar approach to membrane computing via rewriting logic was developed by Lucanu and his collaborators in a series of papers, including [2–5]. The focus in the cited papers was to use rewriting logic to study the existing P-systems, while our approach is more on exploiting the relationship between P-systems and the K framework for enriching each with the strong features of the other.

K was designed as a framework for defining programming languages and has powerful mechanisms to represent programs via its list structures. Our embedding of P-systems in K suggests to include a *control nucleus* in each membrane. The role of this structure it to take care of the rules which are to be used in the membrane. A nucleus generates a set of rules for the next (nondeterministic, parallel maximal) step. When the computation step is finished the rules are deleted and the nucleus produces a new set of rules to be used in the next computation step, and so on. This way, one gets a powerful mechanism

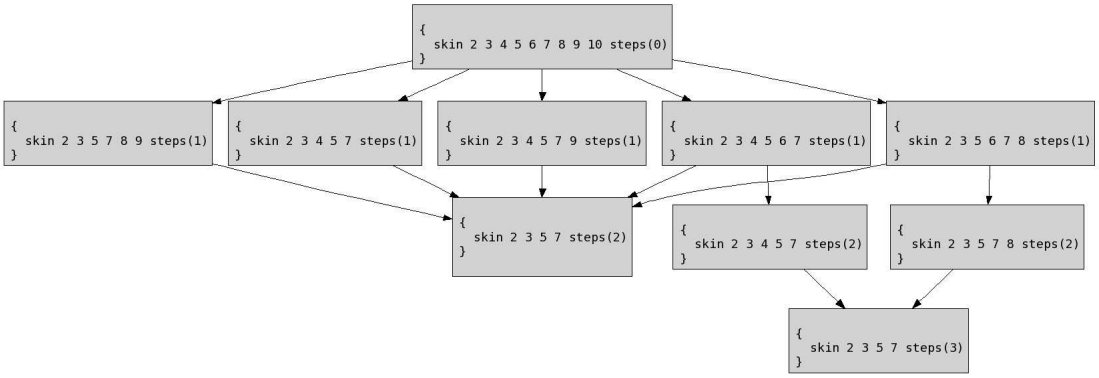


Fig. 7.4 The graph of computing the prime numbers up to 10.

for controlling the evolution of P-systems, narrowing their inherent nondeterminism and opening a way to a better understanding of their behavior.

The classical model of P-systems uses a fixed set of rules for each membrane, so a simple nucleus program may be used to generate this set of rules at each step. One can think at multiple possibilities for these nucleus programs – thanks to the structured objects, any program written in a usual programming language may be used. The embedding of P-systems in K described in this paper naturally extends to this new type of P-systems and may be used to get a running environment for them.

Bibliography

[1] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J.D. Watson: *Molecular Biology of the Cell*, 3rd ed. Garland Publishing, New York, 1994.

[2] O. Andrei, G. Ciobanu, and D. Lucanu: Structural Operational Semantics of P Systems. In: *Proc. Workshop on Membrane Computing 2005*, LNCS 3850, Springer 2006, 31–48.

[3] O. Andrei, G. Ciobanu, and D. Lucanu: Expressing Control Mechanisms of Membranes by Rewriting Strategies. In: *Proc. Workshop on Membrane Computing 2006*, LNCS 4361, Springer 2006: 154–169

[4] O. Andrei, G. Ciobanu, and D. Lucanu: Operational Semantics and Rewriting Logic in Membrane Computing. *Electr. Notes Theor. Comput. Sci.* 156 (2006), 57–78.

[5] O. Andrei, G. Ciobanu, and D. Lucanu: A rewriting logic framework for operational semantics of membrane systems. *Theoretical Computer Science*, 373 (2007), 163–181.

[6] J.-P. Banatre, A. Coutant, and D. Le Metayer: A Parallel Machine for Multiset Transformation and Its Programming Style. *Future Generation Computer Systems*, 4 (1988), 133–144.

[7] G. Berry and G. Boudol: The Chemical Abstract Machine. *Theoretical Computer Science*, 96 (1992), 217–248.
[8] L. Cardelli: Brane Calculi. In: *Proc. Computational Methods in Systems Biology*, LNCS 3082, Springer 2005, 257–278.
[9] G. Ciobanu, G. Paun, G. Stefanescu: P Transducers. *New Generation Computing*, 24 (2006), 1–28.
[10] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, and J. F. Quesada: Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285 (2002), 187–243.
[11] M. Davis, R. Sigal, and E.J. Weyuker. *Computability, Complexity, and Languages.* Second Edition: Fundamentals of Theoretical Computer Science. Morgan Kaufmann, 1994.
[12] P. Frisco. The Conformon-P System: A Molecular and Cell Biology-Inspired Computability Model. *Theoretical Computer Science*, 312 (2004), 295–319.
[13] M. Hills and G. Rosu: KOOL: An Application of Rewriting Logic to Language Prototyping and Analysis. In: *Proc. RTA 2007*, LNCS 4533 Springer 2007, 246–256.
[14] M. Hills, T. Serbanuta, and G. Rosu: A Rewrite Framework for Language Definitions and for Generation of Efficient Interpreters. *Electr. Notes Theor. Comput. Sci*, 176, 4 (2007), 215–231.
[15] J. Meseguer: Conditioned Rewriting Logic as a United Model of Concurrency. *Theoretical Computer Science* 96 (1992), 73–155.
[16] P. Meredith, M. Hills, and G. Rosu: *A K Definition of Scheme.* Technical report UIUCDCS-R-2007-2907, October 2007.
[17] R. Milner: A Theory of Type Polymorphism in Programming. *J. Computer System Sciences* 17(3) (1978), 348–375
[18] R. Milner: *Communication and concurrency*. Prentice-Hall, 1989.
[19] A. Paun and G. Paun: The power of communication: P-systems with symport/antiport. *New Generation Computing*, 20 (2002), 295–306.
[20] G. Paun: Computing with membranes. *Journal of Computer and System Sciences*, 61 (2000), 108–143.
[21] G. Paun: P Systems with Active Membranes: Attacking NP-Complete Problems. *Journal of Automata, Languages and Combinatorics*, 6 (2001), 75–90.
[22] G. Paun: *Introduction to Membrane Computing*. 12th Estonian Winter School in Computer Science, 2007.
[23] G. Rosu: *K: A Rewriting-Based Framework for Computations – Preliminary version.* Technical Report UIUCDCS-R-2007-2926, Department of Computer Science, University of Illinois, 2007. Previous versions published as technical reports UIUCDCS-R-2006-2802 in 2006, UIUCDCS-R-2005-2672 in 2005. K was first introduced in the context of Maude in Fall 2003 as part of a programming language design course (report UIUCDCS-R-2003-2897). `http://fsl.cs.uiuc.edu/k`.
[24] URL: The Web Page of Membrane Computing: http://ppage.psystems.eu/

Translating Multiset Tree Automata into P Systems

José M. Sempere

Universidad Politécnica de Valencia,
Departamento de Sistemas Informáticos y Computación,
Camino de Vera s/n. 46022 Valencia, Spain
jsempere@dsic.upv.es

In this work we propose a translation scheme to obtain P systems with membrane creation and division rules from transitions of Multiset Tree Automata (MTA).

1 Introduction

The relation between membrane structures and Multiset Tree Automata (MTA) has been explored in previous works. So, in [11] we introduced the formal model of Multiset Tree Automata, and in [6] this model was used to calculate editing distances between membrane structures. A method to infer multiset tree automata from membrane observations was presented in [12], while in [13] different families of membrane structures were characterized by using multiset tree automata.

In this work we propose a translation scheme to obtain membrane rules from MTA transitions. The advantages of this approach are clear so we can implement a computer tool to automatically obtain membrane rules (i.e. simple P systems) from a set of trees that model the desired behavior of the membrane structures according to [12]. The structure of this work is the following: first we give basic definitions and notation for tree languages, P systems and multiset tree automata. Then, we propose a translation scheme to obtain membrane rules from MTA transitions. We analyze the correctness and efficiency of the proposed scheme. In the last section, we give some guidelines for future research.

2 Notation and definitions

In the sequel we provide some concepts from formal language theory, membrane systems and multiset processing. We suggest the books [10], [8] and [2] to the reader.

Multisets. First, we provide some definitions from multiset theory as exposed in [14].

Let D be a set. A multiset over D is a pair $\langle D, f\rangle$ where $f : D \longrightarrow \mathbb{N}$ is a function. Suppose that $A = \langle D, f\rangle$ and $B = \langle D, g\rangle$ are two multisets, then the substraction of multiset B from A, denoted by $A \ominus B$, is the multiset $C = \langle D, h\rangle$ where for all $a \in D$ $h(a) = max(f(a) - g(a), 0)$. The sum of A and B is the multiset $C = \langle D, h\rangle$, where for all $a \in D$ $h(a) = f(a) + g(a)$, denoted by $A \oplus B$. We say that A is *empty* if for all $a \in D$, $f(a) = 0$, and $A = B$ if the multiset $(A \ominus B) \oplus (B \ominus A)$ is empty.

The size of a multiset M is the number of elements that it contains and it is denoted by $|M|$ (observe that we take into account the multiplicities of every element). We are specially interested in the class of multisets that we call *bounded multisets*. They are multisets that hold the property that the sum of all the elements is bounded by a constant n. Formally, we denote by $\mathcal{M}_n(D)$ the set of all multisets $\langle D, f\rangle$ such that $\sum_{a\in D} f(a) = n$ (observe that, in this case, $\langle D, f\rangle$ should be finite).

A concept that is quite useful to work with sets and multisets is the *Parikh mapping*. Formally, a Parikh mapping can be viewed as the application $\Psi : D^* \rightarrow \mathbb{N}^n$ where $D = \{d_1, d_2, \cdots, d_n\}$. Given an element $x \in D^*$ we define $\Psi(x) = (\#_{d_1}(x), \cdots, \#_{d_n}(x))$ where $\#_{d_j}(x)$ denotes the number of occurrences of d_j in x, $1 \leq j \leq n$. Finally, in the following, we work with strings representing multisets. So, the multiset represented by x is the multiset with elements that appear in x and multiplicities according to $\Psi(x)$.

P systems. We introduce basic concepts from membrane systems taken from [8]. A transition P system of degree m is a construct

$$\Pi = (V, T, C, \mu, w_1, \cdots, w_m, (R_1, \rho_1), \cdots, (R_m, \rho_m), i_0),$$

where:

- V is an alphabet (the *objects*),
- $T \subseteq V$ (the *output alphabet*),
- $C \subseteq V$, $C \cap T = \emptyset$ (the *catalysts*),
- μ is a membrane structure consisting of m membranes,
- w_i, $1 \leq i \leq m$, is a string representing a multiset over V associated with the region i,
- R_i, $1 \leq i \leq m$, is a finite set of *evolution rules* over V associated with the ith region and ρ_i is a partial order relation over R_i specifying a *priority*.
  An evolution rule is a pair (u, v) (or $u \rightarrow v$) where u is a string over V and $v = v'$ or $v = v'\delta$ where v' is a string over
  $$\{a_{here}, a_{out}, a_{in_j} \,|\, a \in V, 1 \leq j \leq m\}$$
  and δ is a symbol not in V that defines the *membrane dissolving action*. From now on, we denote the set tar by $\{here, out, in_k \,|\, 1 \leq k \leq m\}$,

- i_0 is a number between 1 and m and it specifies the *output* membrane of Π (in the case that it equals to ∞ the output is read outside the system).

The language generated by Π in external mode ($i_0 = \infty$) is denoted by $L(\Pi)$ and it is defined as the set of strings that can be defined by collecting the objects that leave the system by arranging them in the leaving order (if several objects leave the system at the same time then permutations are allowed). The set of vector numbers that represent the objects in the output membrane i_0 is denoted by $N(\Pi)$. Obviously, both sets $L(\Pi)$ and $N(\Pi)$ are defined only for *halting computations*.

Many kinds of rules have been proposed in P systems for creation, division and modification of membrane structures. There have been several works in which these rules have been proposed or employed for different purposes (see, for example, [1, 7–9]).

In the following, we enumerate two kinds of rules which are able to modify the membrane structure, according to [1]

1. Division: $[_h a]_h \rightarrow [_h[_{h_1} a_1]_{h_1}[_{h_2} a_2]_{h_2} \cdots [_{h_p} a_p]_{h_p}]_h$. The object a in region h is transformed into objects $a_1, a_2, \cdots, a_p$. Then, p new regions are created inside h with labels $h_1, h_2, \cdots, h_p$, and the new objects are communicated to the new regions. This rule is a generalization of the 2-division rule proposed in different works such as [1].
2. Creation: $a \rightarrow [_h b]_h$. A new region is created with label h and the object a is transformed into object b which is communicated to the new region.

The power of P systems with the previous operations and other ones (e.g., *exocytosis, endocytosis*, etc.) has been widely studied in the membrane computing area. Given that the previous operations can modify the membrane structure of a P system Π during the computation, we denote by $str(\Pi)$ the set of membrane structures (trees) that eventually are hold by Π during its computation. Observe that this definition was used by Freund *et al.* [4], in order to define tree languages generated by Π systems. In such case, only the membrane structures obtained after halting were considered.

Tree automata and tree languages. Now, we introduce some concepts from tree languages and automata as exposed in [3, 5]. First, let a *ranked alphabet* be the association of an alphabet V together with a finite relation r in $V \times \mathbb{N}$. We denote by V_n the subset $\{\sigma \in V \mid (\sigma, n) \in r\}$. We denote by $maxarity(V)$ the maximum integer k such that $V_k \neq \emptyset$.

The set V^T of trees over V, is defined inductively as follows:

$a \in V^T$ for every $a \in V_0$

$\sigma(t_1, ..., t_n) \in V^T$ whenever $\sigma \in V_n$ and $t_1, ..., t_n \in V^T$, $(n > 0)$

and let a *tree language* over V be defined as a subset of V^T.

Given the tuple $l = \langle 1, 2, ..., k\rangle$ we denote the set of permutations of l by $perm(l)$. Let $t = \sigma(t_1, ..., t_n)$ be a tree over V^T. We denote the set of permutations of t at first level by $perm_1(t)$. Formally, $perm_1(t) = \{\sigma(t_{i_1}, ..., t_{i_n}) \,|\, \langle i_1, i_2, ..., i_n\rangle \in perm(\langle 1, 2, ..., n\rangle)\}$.

Let $\mathbb{N}^*$ be the set of finite strings of natural numbers formed by using the catenation as the composition rule and the empty word λ as the identity. Let the prefix relation $\leq$ in $\mathbb{N}^*$ be defined by the condition that $u \leq v$ if and only if $u \cdot w = v$ for some $w \in \mathbb{N}^*$ $(u, v \in \mathbb{N}^*)$. A finite subset D of $\mathbb{N}^*$ is called a *tree domain* if:

$$u \leq v \text{ where } v \in D \text{ implies } u \in D, \text{ and}$$
$$u \cdot i \in D \text{ whenever } u \cdot j \in D \ (1 \leq i \leq j)$$

Each tree domain D could be seen as an unlabeled tree whose nodes correspond to the elements of D where the hierarchy relation is the prefix order. Thus, each tree t over V can be seen as an application $t : D \to V$. The set D is called the *domain of the tree* t, and denoted by $dom(t)$. The elements of the tree domain $dom(t)$ are called *positions* or *nodes* of the tree t. We denote by $t(x)$ the label of a given node x in $dom(t)$.

Definition 1 *A* deterministic finite tree automaton *is defined by the tuple* $A = (Q, V, \delta, F)$ *where* Q *is a finite set of states;* V *is a ranked alphabet with* m *as the maximum integer in the relation* r*,* $Q \cap V = \emptyset$*;* $F \subseteq Q$ *is the set of final states and* $\delta = \bigcup_{i:V_i \neq \emptyset} \delta_i$ *is a set of transitions defined as follows:*

$$\delta_n : (V_n \times (Q \cup V_0)^n) \to Q \qquad n = 1, \ldots, m$$
$$\delta_0(a) = a \qquad \forall a \in V_0$$

Given the state $q \in Q$, we define the *ancestors* of the state q, denoted by $Ant(q)$, as the set of strings

$$Ant(q) = \{p_1 \cdots p_n \,|\, p_i \in Q \cup V_0 \wedge \delta_n(\sigma, p_1, ..., p_n) = q\}$$

From now on, we refer to deterministic finite tree automata simply as *tree automata*. We suggest [3, 5] for other definitions on tree automata.

The transition function δ is extended to a function $\delta : V^T \rightarrow Q \cup V_0$ on trees as follows:

$$\delta(a) = a \text{ for any } a \in V_0$$
$$\delta(t) = \delta_n(\sigma, \delta(t_1), \ldots, \delta(t_n)) \text{ for } t = \sigma(t_1, \ldots, t_n)\ (n > 0)$$

Note that the symbol δ denotes both the set of transition functions of the automaton and the extension of these functions to operate on trees. In addition, you can observe that the tree automaton A cannot accept any tree of depth zero.

Multiset tree automata and mirrored trees. We extend some definitions of tree automata and tree languages over multisets. We introduce the concept of multiset tree automaton and then we characterize the set of trees that it accepts.

Given any tree automaton $A = (Q, V, \delta, F)$ and $\delta_n(\sigma, p_1, p_2, \ldots, p_n) \in \delta$, we can associate to δ_n the multiset $\langle Q \cup V_0, f\rangle \in \mathcal{M}_n(Q \cup V_0)$ where f is defined by $\Psi(p_1 p_2 \ldots p_n)$. The multiset defined in such way is denoted by $M_\Psi(\delta_n)$. Alternatively, we can define $M_\Psi(\delta_n)$ as $M_\Psi(p_1) \oplus M_\Psi(p_2) \oplus \cdots \oplus M_\Psi(p_n)$ where $\forall 1 \leq i \leq n\ M_\Psi(p_i) \in \mathcal{M}_1(Q \cup V_0)$. Observe that if $\delta_n(\sigma, p_1, p_2, \ldots, p_n) \in \delta$, $\delta'_n(\sigma, p'_1, p'_2, \ldots, p'_n) \in \delta$ and $M_\Psi(\delta_n) = M_\Psi(\delta'_n)$ then δ_n and δ'_n are defined over the same set of states and symbols but in different order (that is the multiset induced by $\langle p_1, p_2, \cdots, p_n\rangle$ equals the one induced by $\langle p'_1 p'_2 \ldots p'_n\rangle$).

Now, we can define a *multiset tree automaton* that performs a bottom-up parsing as in the tree automaton case.

Definition 2 *A* multiset tree automaton *is defined by the tuple* $MA = (Q, V, \delta, F)$*, where* Q *is a finite set of states,* V *is a ranked alphabet with* $maxarity(V) = n$*,* $Q \cap V = \emptyset$*,* $F \subseteq Q$ *is a set of final states and* δ *is a set of transitions defined as follows:*

$$\delta = \bigcup_{\substack{1 \leq i \leq n \\ i\,:\,V_i \neq \emptyset}} \delta_i$$

$$\delta_i : (V_i \times \mathcal{M}_i(Q \cup V_0)) \rightarrow \mathcal{P}(\mathcal{M}_1(Q)) \qquad i = 1, \ldots, n$$
$$\delta_0(a) = M_\Psi(a) \in \mathcal{M}_1(Q \cup V_0) \qquad \forall a \in V_0$$

We can observe that every tree automaton A defines a multiset tree automaton MA as follows

Definition 3 *Let $A = (Q, V, \delta, F)$ be a tree automaton. The multiset tree automaton induced by A is defined by the tuple $MA = (Q, V, \delta', F)$ where each δ' is defined as follows: $M_\Psi(r) \in \delta'_n(\sigma, M)$ if $\delta_n(\sigma, p_1, p_2, ..., p_n) = r$ and $M_\Psi(\delta_n) = M$.*

Observe that, in the general case, the multiset tree automaton induced by A is non deterministic.

As in the case of tree automata, δ' could also be extended to operate on trees. Here, the automaton carries out a bottom-up parsing where the tuples of states and/or symbols are transformed by using the Parikh mapping Ψ to obtain the multisets in $\mathcal{M}_n(Q \cup V_0)$. If the analysis is completed and δ' returns a multiset with at least one final state, the input tree is accepted. So, δ' can be extended as follows

$$\begin{aligned}
&\delta'(a) = M_\Psi(a) \text{ for any } a \in V_0 \\
&\delta'(t) = \{M \in \delta'_n(\sigma, M_1 \oplus \cdots \oplus M_n) \mid M_i \in \delta'(t_i) 1 \leq i \leq n\} \\
&\qquad \text{for } t = \sigma(t_1, \ldots, t_n)\ (n > 0)
\end{aligned}$$

Formally, every multiset tree automaton MA accepts the following language

$$L(MA) = \{t \in V^T \mid M_\Psi(q) \in \delta'(t), q \in F\}$$

Another extension which can be useful is the one related to the ancestors of every state. So, we define $Ant_\Psi(q) = \{M \mid M_\Psi(q) \in \delta_n(\sigma, M)\}$.

The following two results formally relate tree automata and multiset tree automata.

Theorem 1 *(Sempere and López, [11]) Let $A = (Q, V, \delta, F)$ be a tree automaton, $MA = (Q, V, \delta', F)$ be the multiset tree automaton induced by A and $t = \sigma(t_1, \ldots, t_n) \in V^T$. If $\delta(t) = q$ then $M_\Psi(q) \in \delta'(t)$.*

Corollary 1 *(Sempere and López, [11]) Let $A = (Q, V, \delta, F)$ be a tree automaton and $MA = (Q, V, \delta', F)$ be the multiset tree automaton induced by A. If $t \in L(A)$ then $t \in L(MA)$.*

We introduce the concept of *mirroring* in tree structures as exposed in [11]. Informally speaking, two trees are related by mirroring if some permutations at the structural level hold. We propose a definition that relates all the trees with this mirroring property.

Definition 4 *Let t and s be two trees from V^T. We say that t and s are* mirror equivalent, *denoted by $t \bowtie s$, if one of the following conditions holds:*

1. $t = s = a \in V_0$
2. $t \in perm_1(s)$
3. $t = \sigma(t_1, \ldots, t_n)$, $s = \sigma(s_1, \ldots, s_n)$ *and there exists* $\langle s^1, s^2, \ldots, s^k \rangle \in perm(\langle s_1, s_2, ..., s_n \rangle)$ *such that* $\forall 1 \leq i \leq n\ t_i \bowtie s^i$

Theorem 2 *(Sempere and López, [11]) Let $A = (Q, V, \delta, F)$ be a tree automaton, $t = \sigma(t_1, \ldots, t_n) \in V^T$ and $s = \sigma(s_1, \ldots, s_n) \in V^T$. Let $MA = (Q, V, \delta', F)$ be the multiset tree automaton induced by A. If $t \bowtie s$ then $\delta'(t) = \delta'(s)$.*

Corollary 2 *(Sempere and López, [11]) Let $A = (Q, V, \delta, F)$ be a tree automaton, $MA = (Q, V, \delta', F)$ the multiset tree automaton induced by A and $t \in V^T$. If $t \in L(MA)$ then, for any $s \in V^T$ such that $t \bowtie s$, $s \in L(MA)$.*

3 From MTA transitions to membrane rules

In this section, we propose a translation scheme to obtain P systems from MTA. The relation between the input and the output of the scheme is showed at the end of this section. In addition, we show that the obtained P system generates membrane structures which can be represented by trees that the input MTA accepts. First, we provide a couple of examples that give some intuition in the scheme that we propose later.

Example 1 Consider the multiset tree automaton with transitions:

$$\begin{aligned} \delta(\sigma, aa) &= q_1 \\ \delta(\sigma, a) &= q_2 \\ \delta(\sigma, aq_2) &= q_2 \\ \delta(\sigma, q_1 q_1) &= q_1 \\ \delta(\sigma, aq_2 q_1) &= q_3 \in F \end{aligned}$$

Then, the following P system is able to produce, during different computations, a set of membrane structures such that the set of trees induced by them are the set of trees accepted by the MTA.

$\Pi = (\{a, b\}, \{a, b\}, \emptyset, []_{q_3}, b, \emptyset, \emptyset, (R_{q_3}, \emptyset), (R_{q_2}, \emptyset), (R_{q_1}, \emptyset), \infty)$, where

$R_{q_3} = \{b \rightarrow a[_{q_2} b]_{q_2} [_{q_1} b]_{q_1}\}$,

$R_{q_2} = \{b \to a[_{q_2}b]_{q_2}; b \to a\}$, and

$R_{q_1} = \{b \to aa; b \to [_{q_1}b]_{q_1}[_{q_1}b]_{q_1}\}$

Observe that we have made a top-down design, in which we start by analyzing the final states (in this case q_3) and then we obtain the ancestors of every state according with δ by using membrane creation and membrane division.

Let us see the following example, where the number of final states is greater than one.

Example 2 Let us take the MTA defined by the following transitions

$$\begin{aligned} \delta(\sigma, aa) &= q_1 \in F && (1)\\ \delta(\sigma, bb) &= q_2 \in F && (2)\\ \delta(\sigma, q_2q_2) &= q_2 \in F && (3)\\ \delta(\sigma, q_1q_1) &= q_1 \in F && (4)\\ \delta(\sigma, q_2q_1) &= q_3 \in F && (5) \end{aligned}$$

The following P system is associated to the previous MTA. Observe that we have added a superscript to every membrane rule according to every MTA transition.

$\Pi = (\{a, b, c\}, \{a, b, c\}, \emptyset, []_0, c, \emptyset, \emptyset, \emptyset, (R_0, \emptyset), (R_{q_2}, \emptyset), (R_{q_1}, \emptyset), \infty)$, where

$$\begin{aligned} R_0 = \quad & \{c \to aa^{(1)}; c \to bb^{(2)}; c \to [_{q_2}c]_{q_2}[_{q_2}c]^{(3)}_{q_2}; c \to [_{q_1}c]_{q_1}[_{q_1}c]^{(4)}_{q_1};\\ & c \to [_{q_2}c]_{q_2}[_{q_1}c]^{(5)}_{q_1}\}\\ R_{q_2} = \quad & \{c \to bb^{(2)}; c \to [_{q_2}c]_{q_2}[_{q_2}c]^{(3)}_{q_2}\}\\ R_{q_1} = \quad & \{c \to aa^{(1)}; c \to [_{q_1}c]_{q_1}[_{q_1}c]^{(4)}_{q_1}\} \end{aligned}$$

We propose Algorithm 1 as a translation scheme from MTA to P systems. The main step of the proposed algorithm, is step 5 which uses a transformation $\wp_c$ over $M_\Psi(\delta)$. We formally define the transformation $\wp_c$ as follows: $\wp_c(p_1 \cdots p_k) = \wp_c(p_1) \cdots \wp_c(p_k)$, where

$$\wp_c(p_i) = \begin{cases} p_i & \text{if } p_i \in \Sigma_0\\ [_{p_i}c]_{p_i} & \text{if } p_i \in Q \end{cases}$$

Now, we can formally prove the correctness of the proposed algorithm through the following result.

Algorithm 1 A translation scheme from MTA to P systems.

Input: A MTA $A = (Q, \Sigma, \delta, F)$
Output: A P system $\Pi = (V, T, \emptyset, []_0, c, \emptyset, \cdots, \emptyset, (R_0, \rho_0), \cdots, (R_m, \rho_m), i_0)$ such that $str(\Pi) = L(A)$
Method:

1. $V = T = \Sigma_0 \cup \{c\}$ such that $c \notin \Sigma_0$
2. $m = |Q|$
3. $\rho_i = \emptyset\ 0 \leq i \leq |Q|$
4. $R_i = \emptyset\ 0 \leq i \leq |Q|$
5. For every transition in δ such that $\delta(\sigma, p_1 \cdots p_k) = q_j$
   If $q_j \in F$
   then Add to R_0 the rule $c \rightarrow \wp_c(p_1 \cdots p_k)$
   Add to R_j the rule $c \rightarrow \wp_c(p_1 \cdots p_k)$
6. **Return(**Π**)**

EndMethod.

Proposition 1 *Algorithm 1 obtains a P system* Π *from the input MTA* A *such that* $str(\Pi) = L(A)$.

Proof The key step in the proposed algorithm is step 5. Observe that the step 5, ensures that if $\delta(\sigma, p_1 \cdots p_k) = q_j$ then the region R_j holds a rule such that for every state p_l in the ancestors of q_j, according to the transition, a new region $[_{q_l} c]_{q_l}$ is created. On the other hand, every symbol $a \in p_1 \cdots p_k$ is created in region R_j. So, if the structure $\sigma(p_1 \cdots p_k)$ (or any mirrored one) is reduced to the state q_j in the MTA A, the structure $\wp_c(p_1 \cdots p_k)$ is created in the P system Π inside the region R_j. In addition, if $q_j \in F$ then all the (mirrored) trees reduced to q_j are accepted by A, so this is the reason why all these structures are inside the skin region R_0.

On the other hand, observe that the unique object which can create new membranes is c which does not belong to Σ_0. We have introduced c because the rest of symbols are just leaves in the trees accepted by A. So, once any of the leaves appears, it remains in the region as a an object that cannot evolve anymore. Finally, the objects c disappear when all the leaves of the trees are created.

□

Another aspect that we take under our consideration is the efficiency of the proposed algorithm. We analyze its complexity time through the following result.

Proposition 2 *Algorithm 1 runs in polynomial time with respect to the size of the input MTA* A.

Proof Again, the main step of the proposed algorithm is step 5. Here, we make as many operations as the number of δ transitions. For every transition, we must evaluate the transformation $\wp_c$ which is quadratic with the size of the ancestors of every state and the union of $|Q|$ and Σ_0. This holds a quadratic running time for Algorithm 1. □

4 Conclusions and future work

In this work we have proposed a full translation scheme from MTA to P systems. The proposed algorithm correctly and efficiently performs the translation task. This scheme gives a formal proof for the relation between the structures generated by the P system with membrane creation and membrane division and the trees accepted by MTA. This result was pointed out in previous works such as [6, 11–13].

Actually, we are developing a computer tool that holds the proposed translation scheme. This tool will help to analyze the membrane dynamics in P systems by using the results proposed in [12]. Furthermore, we will be able to propose initial P systems based only in the membrane structures we want to generate which will be enriched later with the corresponding evolution and communication rules.

On the other hand, a topic which has been investigated in previous works is the relationship between MTA and P systems. We can study in depth some aspects of the P systems by only observing the membrane dynamics. This study can be achieved by characterizing different MTA classes as was proposed in [13]. We think that we must keep on this research in order to get a complex picture of different P systems and their relations by using only MTA.

Acknowledgements. Work supported by the Spanish Ministerio de Educación y Ciencia under project TIN2007-60769.

The author is grateful to the reviewers for sharp remarks and suggestions made to this work.

Bibliography

[1] A. Alhazov, T.O. Ishdorj. *Membrane operations in P systems with active membranes.* In Proc. Second Brainstorming Week on Membrane Computing. TR 01/04 of RGNC. Sevilla University. pp 37-44. 2004.

[2] C. Calude, Gh. Păun, G. Rozenberg and A. Salomaa (Eds.), *Multiset Processing* LNCS 2235. Springer. 2001.

[3] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree automata techniques and applications.* Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997. release October, 1rst 2002.

[4] R. Freund, M. Oswald, A. Păun. *P systems generating trees.* In Proceedings of the 5th International Workshop on Membrane Computing, WMC 2004, pp 309-319. G. Mauri, Gh. Păun, M. Pérez-Jiménez, G. Rozenberg, A. Salomaa (Eds.) LNCS 3365, Springer. 2005.

[5] F. Gécseg and M. Steinby. *Handbook of Formal Languages*, volume 3, chapter Tree languages, pages 1–69. Springer-Verlag, 1997.

[6] D. López, J. M. Sempere. *Editing distances between membrane structures*. In Proceedings of the 6th International Workshop, WMC 2005, pp 326-341. R. Freund, Gh. Păun, G. Rozenberg and A. Salomaa (Eds.). LNCS 3850, Springer. 2006.

[7] A. Păun. *On P systems with active membranes.* In Proceedings of the Second International Conference on Unconventional Models of Computation (UMC'2K). pp 187-201. I. Antoniou, C.S. Calude and M.J. Dinnen (Eds.). Springer. 2001.

[8] Gh. Păun. *Membrane Computing. An Introduction.* Springer. 2002.

[9] Gh. Păun, Y. Suzuki, H. Tanaka, T. Yokomori. *On the power of membrane division on P systems.* Theoretical Computer Science 324, 1 pp 61–85. 2004.

[10] G. Rozenberg, A. Salomaa (Eds.). *Handbook of Formal Languages* Vol. 1. Springer. 1997.

[11] J. M. Sempere, D. Lpez. *Recognizing membrane structures with tree automata.* In 3rd Brainstorming Week on Membrane Computing 2005. RGNC Report 01/2005 Research Group on Natural Computing. Sevilla University (M.A. Gutirrez Naranjo, A. Riscos-Nez, F.J. Romero-Campero, D. Sburlan, eds.) pp. 305-316. Fnix Editora 2005.

[12] J. M. Sempere, D. López. *Identifying P rules from membrane structures with an error-correcting approach.* In Proceedings of the 7th International Workshop WMC 2006, pp 507-520. H. Jan Hoogeboom, Gh. Păun and G. Rozenberg (Eds.). LNCS 4361, Springer-Verlag. 2006

[13] J. M. Sempere, D. López. *Characterizing membrane structures through multiset Tree Automata* In Proceedings of the 8th Workshop on Membrane Computing (WMC8), pp 428-437. G. Eleftherakis, P. Kefalas, Gh. Păun (Eds.). LNCS 4860, Springer. 2007.

[14] A. Syropoulos. *Mathematics of multisets.* In [2] pp 347-358.

Chemical reaction simulations using Abstract Rewriting System on Multisets with Lattice Boltzmann Method

Mai Umeki, Yasuhiro Suzuki

Nagoya University, Graduate School of Information Science,
Furocho Chikusa Nagoya City, 464-8603, Japan

We have proposed a deterministic approach for simulating chemical reactions, the Deterministic Abstract Rewriting Systems on Multisets, DARMS. DARMS is an approximate procedure of an exact stochastic method of simulating chemical reactions. DARMS is not a stochastic method but a deterministic method, where reactions rules are applied in maximally parallel. Thus DARMS has good accordance with conventional P Systems with acceptable loss of chemical accuracy. We have composed a simulation method for the reaction-diffusion-convection model of chemical reactions by synthesizing DARMS and the Lattice Boltzmann Method (LBM); LBM is a discrete expression of the Navier-Stokes Equation.

1 Introduction

DARMS, a variant of conventional P System with chemical accuracy. There are two formalisms for simulating spatially homogeneous chemical system; the deterministic approach by a set of differential equations and the stochastic approach based on a single differential equation. The stochastic approach has a firmer physical basis, which is based on the master equation than the deterministic approach, but the master equation is often mathematically intractable. Thus there have proposed a method to make exact numerical calculations within the framework of the stochastic formulation, such as the Gillespie method, without having to deal with the master equation directly. However such method requires large amount of system time on simulation. And in P Systems, for simulating (bio)chemical systems, the MP system has been known [13] [3]. A novel method that we propose is Deterministic Abstract Rewriting System on Multisets (DARMS), which is a deterministic approach based on an approximate procedure of an exact stochastic method. DARMS can produce significant gains in simulation speed with acceptable losses in accuracy and since reaction rules are applied deterministically (not stochastic) and maximally parallel so it has good accordance with conventional P Systems [16] with acceptable loss of chemical accuracy.

Reaction-Diffusion-Convection model. Recently, reaction-diffusion-convection (RDC) phenomenon, RDC in a nonequilibrium chemical reaction have been observed with interest by reseachers. For example, the Belousov-Zhabotinskii reaction (BZ reaction) is a typical nonequilibrium chemical reaction but also the RDC can be observed. In order to understand nonequilibrium phenomena, it is important to understand the reaction-diffusion-convection phenomenon, however in order to model the RDC, we have to integrate the reaction-diffusion equation and the Navier-Stokes equation and it is not easy to simulate and analyze. Thus in order to model the RDC, we integrate the DARMS and the Lattice Boltzmann Method, LBM. LBM is a discrete expression of the Navier-Stokes equation.

2 Abstract Rewriting System on Multisets, ARMS

An ARMS [17] is a construct $\Gamma = (A, w, R)$, where A is an alphabet, w is a multiset present in the initial configuration of the system, and R is the set of multiset rewriting rules.

Let A be an *alphabet* (a finite set of abstract symbols). A *multiset* over A is a mapping $M : A \mapsto \mathbf{N}$, where $\mathbf{N}$ is the set of natural numbers; 0, 1, 2,.... For each $a_i \in A$, $M(a_i)$ is the *multiplicity* of a_i in M, we also denote $M(a_i)$ as $[a_i]$.

We denote by $A^{\#}$ the set of all multisets over A, with the empty multiset, $\emptyset$, defined by $\emptyset(a) = 0$ for all $a \in A$.

A multiset $M : A \mapsto \mathbf{N}$, for $A = \{a_1, \ldots, a_n\}$ is represented by the state vector $w = (M(a_1), M(a_2), \ldots, M(a_n))$, w. The union of two multisets $M_1, M_2 : A \mapsto \mathbf{N}$ is the addition of vectors w_1 and w_2 that represent the multisets M_1, M_2, respectively. If $M_1(a) \leq M_2(a)$ for all $a \in A$, then we say that multiset M_1 is included in multiset M_2 and we write $M_1 \subseteq M_2$.

A *reaction rule r* over A can be defined as a couple of multisets, (s, u), with $s, u \in A^{\#}$. A set of reaction rules is expressed as R. A rule $r = (s, u)$ is also represented as $r = s \rightarrow u$. Given a multiset $s \subseteq$, the application of a rule $r = s \rightarrow u$ to the multiset w produces a multiset w' such that $w' = w - s + u$. Note that s and u can also be zero vector (empty).

The *reaction vector*, ν_{ji} denotes the change of the number of a_i molecules produced by one reaction of rule r_j.

2.1 ARMS with chemical kinetics We modify the ARMS [21] [20] for modeling chemical kinetics and this enables us to use experimentally obtained reaction rates directly, similar to the derivation of the Gillespie's "τ-leap method" [10].

In order to handle experimental data, we employ multisets with real multiplicities; such a multiset $X : A \mapsto \mathbf{R}$ for $A = \{a_1, \ldots, a_n\}$ is represented by the state vector $\mathbf{x} = (X(a_1), X(a_2), \ldots, X(a_n))$. $X(a_i)$ denotes the molar concentration of specie a_i.

Let us assume that there are $N \geq 1$ molecular species $\{a_1, ..., a_n\}, a_i \in A$ that interact through reaction rules $R = \{r_1, ..., r_m\}$. As the time evolution of $\mathbf{x}$ unfolds from a certain initial state, let us suppose the state transition of the system to be recorded by marking on a time axis the successive instants $t_1, t_2, ...$ as $X(t_j)$ $(j = 1, 2, ...)$. We specify the dynamical state of $\mathbf{x}(t) \equiv (X(a_1(t), X(a_2(t)), ..., X(a_N(t)))$, where $X(a_i(t))$ is the molar concentration of a_i specie at time t, $t \in \mathbf{R}$.

Chemical kinetics. We assume that all chemical reactions take place in a well-stirred reactor; this assumption is required due to the strong dependence of the reaction rate on the concentration of the reagent species. We define the function f_j, called the *propensity function* for $r_j \in R$ by

$$f_j(x) = c_j h_j, \tag{51}$$

where c_j denotes the average probability that a particular combination of r_j reactant molecules will react in the next infinitesimal time interval dt and h_j is the number of possible combinations of the species of r_j in dt.

$f_j \mathbf{x}(t)dt$ means that the probability that reaction r_j will occur in the next infinitesimal time interval $[t, t + dt),\ (j = 1, ..., m)$.

The time evolution of $\mathbf{x}(t)$ is a jump Markov process [12] on the N-dimensional non-negative lattice. In this case, an ARMS has a *macroscopically infinitesimal time scale*, Δ, where reaction rules can be applied several times simultaneously, yet since the stoichiometrical change of the state during Δ is small enough, none of the propensity functions change appreciably.

The parameter Δ corresponds to τ (small time interval) in the Gillespie's method [10] and it satisfies the *Leap Condition* given below; an amount Δ that spans a *very large* number of applying every reaction rules *still* satisfies the Leap Condition.

Leap Condition: We require Δ to be small enough that the change in the state during $[t, t + \Delta]$ will be so small that no propensity function will suffer an appreciable (i.e., macroscopically noninfinitesimal) change in its value.

We also assume that the number of applications of each reaction rule in Δ obeys

$$\langle P(f_j(\mathbf{x}), \Delta\rangle = f_j(\mathbf{x})\Delta \gg 1 (\forall j = 1, ..., m), \tag{52}$$

where $P(f_j(\mathbf{x}), \Delta)$ is the *Poisson* random variables is the number of reactions that occur in Δ.

Here, let us consider the probability function *Q*, defined by $Q(z_1, ..., z_k|\Delta, \mathbf{x}, t)$, which means the probability, given **X**(t) = **x**, that in the time interval $[t, t + \delta)$ exactly z_j times of rule applications or r_j will occur, for each $j = 1, ..., m$. *Q* is evidently the joint probability density function of the *M* integer random variables, $Z_j(\Delta, \mathbf{x}, t)$ means the number of times, given **X(t)=x**, that reaction rule r_j will apply in the time interval $[t, t + \Delta)$ $(j = 1, ..., m)$.

If the equation (52) is satisfied, the *Poisson* random numbers will be practically indistinguishable from *normal* random numbers, which are uncorrelated statistically independent normal random variables with mean 0 and variance 1.

Then the jump Markov process can be approximated by the *continuous* Markov process defined by the standard form of *chemical Langevin equation* (CLE).

$$\begin{aligned}\lambda_i = \sum_{j=1}^{m} z_j \nu_{ij} = \sum_{j=1}^{m} f_j \nu_{ji} &= \sum_{j=1}^{m} [f_j(\mathbf{x})\Delta + (f_j(\mathbf{x})\Delta)^{\frac{1}{2}} n_j]\nu_{ji} \\ &= \sum_{j=1}^{m} \nu_{ji} f_j(\mathbf{x})\Delta + \sum_{j=1}^{m} \nu_{ji} f_j^{\frac{1}{2}}(\mathbf{x}) n_j \Delta^{\frac{1}{2}}, \end{aligned} \tag{53}$$

where n_j is temporally uncorrelated statistically independent normal random variables. Since $Z_j(\Delta, \mathbf{x}, t) = P(f_j(\mathbf{x}, \Delta))$, it is equal to $f_j(\mathbf{x})\Delta$, by the equation (52).

In case $f_j(\mathbf{x})\Delta \to \infty$, (52) implies that in the part $f_j(\mathbf{x})\Delta + (f_j(\mathbf{x}\Delta)^{\frac{1}{2}} n_j$ of the equation (53) the second term becomes negligibly small compared to the first term and λ_i in the limit $(f_j(\mathbf{x})\Delta \to \infty)$, because

$$\begin{aligned}\lambda_i = \sum_{j=1}^{m} z_j \nu_{ji} &= \sum_{j=1}^{m} [f_j(\mathbf{x})\Delta]\nu_{ji} \\ &= \sum_{j=1}^{m} \nu_{ji} f_j(\mathbf{x})\Delta. \end{aligned} \tag{54}$$

This is the Euler formula (piecewise linear approximation) for numerically solving the RRE. It shows how to derive the continuous and deterministic RRE of traditional chemical kinetics from the stochastic method. Since $\nu_{ji} f_j(\mathbf{x})$ represents the stoichiometric change in the next infinitesimal time, it can be regarded as the reaction rate of r_j, v_j, and we obtain:

$$\lambda_i = \sum_{j=1}^{m} \nu_{ji} f_j(\mathbf{x})\Delta \equiv \sum_{j=1}^{m} v_j(\mathbf{x})\Delta. \tag{55}$$

In the Gillespie τ leap method, the number of applications of each rule within τ is randomly generated according to the *Poisson* or *Normal* distribution and λ_i is calculated.

In the ARMS, λ_i is calculated by using the reaction rate given by the equation (55). As in the numerically solving an ordinary differential equation of the form $dX/dt = f(X)$ by the Euler method, a leap down the stepwise time axis by Δ according to $X(t+\Delta) = X(t) + f(X(t))\Delta$ will produce errors whenever the function f changes during that Δ increment.

It is well-known that the second-order Runge-Kutta procedure can reduce these errors; use the simple Euler method to estimate the "midpoint" value of X during Δ, and then calculate the actual increment in X by evaluating the slope function f at that estimated midpoint. The midpoint value can be obtained from the expected state change λ as $\mathbf{x} + \frac{\lambda}{2}$. In the Gillespie's τ leap method, this procedure is used and it shows that this procedure can reduce numerical errors [10].

2.2 Algorithm of DARMS In Deterministic Abstract Rewriting System on multisets (DARMS), reaction rules are applied in maximally parallel and deterministic way. Hence, the DARMS accommodates P Systems, while it has background in theoretical chemistry [20].

Step 0(Initialization). The time t is set to 0 and the set of vectors $V = (\delta_1, \delta_2, ..., \delta_N)$ $(j = 1, 2, ..., m)$, expressing the stoichiometric change of each species, are initialized. Then all inputs of the system are assigned to their respective variables,

- $X(a_1), X(a_2), ..., X(a_N)$ are set to the initial quantities of species;
- $k_1, ..., k_m$ to set m rate constants corresponding to the m reactions;
- t_{stop} to the ending instant of simulation;
- set the value of Δ;

Step 1(Calculation of state change vector Λ_t). According to reaction rules, stoichiometric change of each specie λ_i is calculated as well as the state change vector; $\Lambda_t = (\lambda_1, \lambda_2, ..., \lambda_N)$ is calculated, where $\lambda_i = \sum_{j=1}^{m} \nu_{ji} v_j \mathbf{x}(t)\Delta$.

Step 2(System update and branching). The quantity of each species and t is updated, by using Λ_t and Δ:

$$\begin{aligned} \mathbf{x}(t) &= \mathbf{x}(t-\Delta) + \Lambda_{t-\Delta}, \\ t &:= t + \Delta. \end{aligned}$$

If $t \geq t_{stop}$ or if there are no reactions left in the reactor, the simulation is stopped and the results are sent to the output stream. Otherwise, the simulation returns to *Step 1*.

In order to simulate pattern formation, we compose cellular automata by using the DARMS and call it Cellular Automata of Abstract Rewriting System on Multisets (CARMS) [20]. As for the calculation of diffusion, we use conventional explicit scheme of difference method to solve partial differential equation of diffusion and for the calculation of convection, we use the Lattice Boltzmann Method [11].

2.3 Lattice Boltzmann Method (LBM) The lattice Boltzmann equation (LBE) method is emerging as a physically accurate and computationally viable tool for simulating laminar and turbulent flows. On the theoretical front, rigorous mathematical proof now exists demonstrating that the lattice Boltzmann method (LBM) is a special finite difference scheme of the Boltzmann equation that governs all fluid flows (the Navier-Stokes equation also has its basis in the Boltzmann equation).

The basic LBE for a single-component medium consists of two basic steps: collision and advection. The particle distribution function is thermalized locally through collision processes and advection to the closest neighboring sites occurs according to a small set of discrete particle velocities. The LBE proposed here is the lattice Boltzmann scheme with BGK approximation [2];

$$n_\alpha(\mathbf{x} + e_\alpha \delta_t, t + \delta_t) = n_\alpha(\mathbf{x}, t) - \frac{1}{\tau}[n_\alpha(\mathbf{x}, t) - n_\alpha^{(eq)}(\mathbf{x}, t)] \tag{56}$$

where n_α is the number density distribution function with discrete velocity e_α, $n_\alpha^{(eq)}$ is the equilibrium distribution function and τ is the relaxation time (towards equilibrium) which determines the viscosity. The time-step size is δ_t, which is the time taken for the advection process to be completed. For the sake of simplicity without losing generality, we adopt the nine-velocity model. Then the equilibrium distribution function for isothermal field is given as

$$n_\alpha^{(eq)} = w_\alpha n[1 + \frac{1}{c_s^2}(e_\alpha \cdot \mathbf{u}) \times \frac{1}{2c_s^4}(e_\alpha \cdot \mathbf{u})^2 - \frac{1}{c_s^2}u^2] \tag{57}$$

in which the discrete particle velocities e_α and the weighting factor w_α (α = 0,1,2,$\cdots$,8) are

$$e_\alpha = \begin{cases} (0,0) & \alpha = 0 \\ (cos[(\alpha-1)\pi/2]), sin[(\alpha-1)\pi/2] & \alpha = 1,2,3,4 \\ (cos[(\alpha-4)\pi/4]), sin[(\alpha-5)\pi/2+\pi/4] & \alpha = 5,6,7,8 \end{cases} \tag{58}$$

and

$$e_\alpha = \begin{cases} 4/9 & \alpha = 0 \\ 1/9 & \alpha = 1,2,3,4 \\ 1/36 & \alpha = 5,6,7,8 \end{cases} \tag{59}$$

respectively. The sound speed is $w_\alpha = 1/\sqrt{3}(\delta_x/\delta_t)$ with δ_x being the lattice constant of the underlying square lattice. The macroscopic quantities, such as particle density n, mass density ?? and mass velocity $\mathbf{u}$ are given by

$$n = \sum_\alpha n_\alpha \tag{60}$$

$$\rho = mn \tag{61}$$

$$\rho \mathbf{u} = m \sum n_\alpha e_\alpha \tag{62}$$

where m is the molecular weight (for more detail of the LBM, refer [11]).

3 Lattice Boltzman Equations for Reaction flow

In a reacting flow, the state of the fluid at any given point in space and time can be completely specified in terms of fluid velocity, composition vector (either in terms of mass fraction or concentration). We will need to develop the LBE for all these variables. For generating a background flow, the conventional LBM sub-steps of collision (relaxation) and streaming (convection) are used. However for the concentration fields, there is an extra sub-step between collision and streaming sub-steps to account for reaction-diffusion and convection. This is identical to the time-splitting approach used in continuum methods for chemically reacting flows.

The background flow-field is obtained using the following stencil for partial pressure

$$p_\alpha(\mathbf{x} + e_\alpha, t+1) = p_\alpha(\mathbf{x}, t) - \frac{1}{\tau_p}[p_\alpha(\mathbf{x}, t) - p_\alpha^{(eq)(\mathbf{x},t)}] \tag{63}$$

where

$$p_\alpha^{(eq)} = w_\alpha p[1 + 3(e_\alpha \cdot \mathbf{u}) + \frac{9}{2}(e_\alpha \cdot \mathbf{u})^2 - \frac{3}{2}u^2] \tag{64}$$

The total pressure $p(= \rho c_s^2)$ and the fluid velocity are calculated using

$$p = \sum_{\alpha} p_{\alpha} \tag{65}$$

$$u = \frac{1}{p} \sum_{\alpha} e_{alpha} p_{\alpha} \tag{66}$$

This is the velocity used for determining the equilibrium distribution functions in temperature and concentration fields.

3.4 Concentraton fields For concentration field, there is an extra computational sub-step, reaction and diffusion by using the DARMS and CARMS besides conventional computational sub-steps of collision and advection.

Collision of chemical specie i.

$$Y_{\alpha}^{i}(\mathbf{x}, t) = Y_{\alpha}^{i}(\mathbf{x}, t) - \frac{1}{\tau_i}[Y_{\alpha}^{i}(\mathbf{x}, t) - Y_{\alpha}^{i(eq)(\mathbf{x},t)}] \tag{67}$$

where Y^i denotes the concentration of chemical specie i,

$$Y_{\alpha}^{i(eq)} = w_{\alpha} Y^i [1 + 3(e_{\alpha} \cdot u) + \frac{9}{2}(e_{\alpha} \cdot) u)^2 - \frac{3}{2} u^2] \tag{68}$$

and

$$Y^i = \sum_{\alpha} Y_{\alpha}^{i}, \tag{69}$$

Relaxation time-constant τ is determined by thermal diffusivity and τ_i's are determined by the diffusivity of corresponding species.

4 Simulation of the Oregonator

The Oregonator scheme is outlined in Table 4.1: In this paper, a combination of Tyson's "Lo" [19] and Field-Főrsterling values [7] (TFF parameter) are used [14]: $k_1 : 10^6 M^{-2} S^{-1}, k_2 : 2M^{-3}S^{-1}, k_3 : 2 \times 10^3 M^{-1} S^{-1}, k_4 : 10 M^{-2} S^{-1}, k_5 : B \times 2 \times 10^{-2} S^{-1}$, where M stands for one molar, and S stands for a second.

4.5 Results of the simulation We take the non-slip boundary condition (the velocities of particles which hit the wall are inverted after the collision). The condition of the simulation is described as follows; the amount of computation steps is 20,000, $\Delta = 0.01$, $\tau = 10, 1.0 \times 10^4, 1.0 times 10^7$, the diffusion constants D obtained by

$$
\begin{array}{lll}
X, Y, H & \xrightarrow{k_1} & 2W : (r_1), \\
A, Y, 2H & \xrightarrow{k_2} & X, W : (r_2), \\
2X & \xrightarrow{k_3} & A, W, H : (r_3), \\
A, X, H & \xrightarrow{k_4} & 2X, 2Z : (r_4), \\
B, Z & \xrightarrow{k_5} & 0.5Y : (r_5).
\end{array}
$$

Table 4.1 Oregonator

chemical experiments [14]; (cm^2 / sec.) of X, D_X and Z, D_Z are 1.5×10^{-5} and $D_X = 0.9 \times 10^{-5}$.

It is assumed that the size of reactor in the CARMS is a 6cm $\times$ 6cm square, where 50 $\times$ 50 DARMSes are placed. So, the distance between DARMSes is $\Delta x = \frac{6}{50}$cm. In the chemical experiment of BZ reaction, usually a excitation point is generated by stinging a sliver stick, which evokes oxidation reaction. In order to express the generation of the excitation point, we change the concentration of X and Y are smaller, while that of Z is 100 times larger.

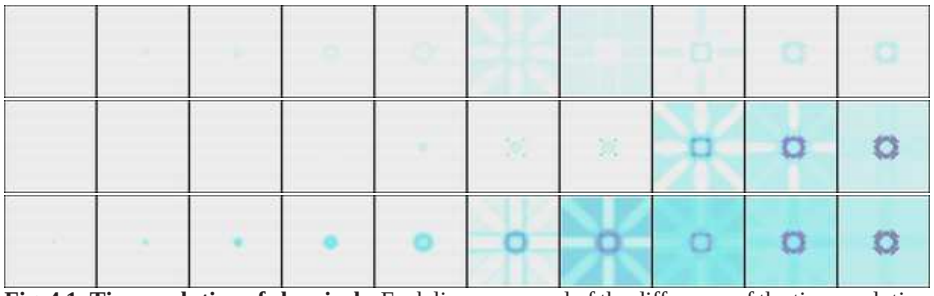


Fig. 4.1 Time evolution of chemicals: Each line composed of the difference of the time evolution of concentration of X (top), Y (middle) and Z (bottom) in the CARMS, where, time evolution starts from right toward left. Blue illustrates that the concentration is high, while white, low and $\tau = 1.0 \times 10^4$

The results of simulation of the Oregonator illustrate that the CARMS with reaction, diffusion and convection exhibits typical chemical wave spatial pattern of the Oregonator on every chemical specie X, Y and Z.

Next, we change effectiveness of the convection. Since the value of τ denotes the effectiveness, we change $\tau = 10$ (the effectiveness is strong), $\tau = 1.0 \times 10^4$ (middle) and $\tau = 1.0 \times 10^7$ (weak). And we confirmed that the effectiveness of the convection change the spatio-temporal pattern of chemical reaction (figure 4.2). When the effectiveness is strong (the top line in the figure 4.2), since the convection was strong, the reactor was well stirred and spatial patterns were excluded, but temporal patterns were preserved. And when the effectiveness is middle (the middle line in the figure), there

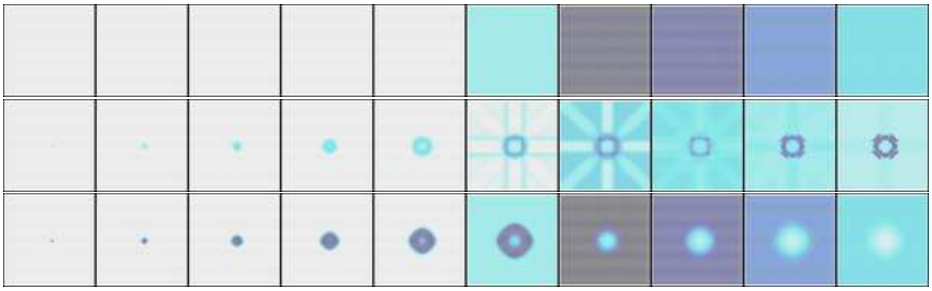


Fig. 4.2 Effectiveness of the convection The difference of the time evolution of concentration of Z in the CARMS, where time evolution starts from right toward left. Blue illustrates that the concentration is high, while white, low. The effect of convection is changed; the value of τ denotes the degree of effectiveness of the convection, as the τ is getting large, the effectiveness becomes large. Each line illustrates when $\tau = 10$ (top), $\tau = 1.0 \times 10^4$ (middle), $\tau = 1.0 \times 10^7$ (bottom), respectively

emerged spatio-temporal pattern, however, its pattern was different from the case when the effectiveness is weak. When the effectiveness of convection is weak, it is almost same to the system only with reaction and diffusion. We confirmed that when the effectiveness of convection is weak, its pattern (the bottom line in the figure) is similar to the ARMS with reaction-diffusion.

Bibliography

[1] S. Barkin, M. Bixon, R.M. Noyes and K. Bar-Eli, The oxidation of cerous ions by bromate ions comparison of experimental data with computer calculations, Int. J. Chem. Kinet. 9: 841-862. 1977.

[2] Bhatnagar, P.L., Gross, E.P., Krook, K. (1954). A model for collision processes in gases. Phys. Rev. 94: 511.524.

[3] L. Bianco, Membrane Models of Biological Systems, Ph.D. Thesis, Univ. of Verona, 2007.

[4] R.J. Field, E. Kőrős and R.M. Noyes, Oscillation in chemical systems II, Through analysis of temporal oscillation in the bromate-cerium-malonic acid system, J. Am. Chem. Soc. 94, 8649-8664, 1972.

[5] R.J. Field and R.M. Noyes, Oscillations in chemical systems. IV. Limit cycle behavior in a model of a real chemical reaction, J. Chem. Phys. 60, 5, 1, 1877-1884, 1974.

[6] R.J. Field and M. Burger, Oscillation and Traveling Waves in Chemical Systems, Willey, New York, 1985.

[7] R.J. Field and Horst-Dieter F0̋rsteling, On the oxybromine chemistry rate constants with cerium ion in the Field-Kőoős-Noyes mechanism of the Belousov-Zhabotinskii reaction, J. Phys. Chem. 90, 5400-5407, 1986.

[8] D.T. Gillespie, A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions, J. Comp. Phys., 22:403-434, 1976.

[9] D.T. Gillespie, Exact Stochastic Simulation of Coupled Chemical Reactions, J. Phys. Chem. 81 (25), 2340-2361, 1977.

[10] D.T. Gillespie, Approximate accelerated stochastic simulation of chemically reacting systems, J. Chem. Phys, 115(4), 1716-1733, 2001.

[11] Huidan Y., L.S. Luo, S. S. Girimaji, Scalar mixinig and chemical reaction simulations using lattice boltzmann method, International Journal of Computational Engineering Science Vol. 3, No. 1 (2002) 73-87, Imperial College Press. Boltzmann Method, J. Comput. Phys. 118, 329-347 1995.

[12] J. Jacod, A.V. Skorokhod, Jumping Markov Processes. Annales de l'institut Henri Poincare (B) Probabilites et Statistiques, 32 no. 1 (1996), p. 11-67

[13] V. Manca, String rewriting and metabolism: a logical perspective, in Computing with Bio-Molecules. Theory and Experiments, 36-60, Springer Verlag, Singapore, 1998.

[14] H. Miike, Y. Mori and T. Yamaguchi, Science of non-equilibrium Systems (in Japanese), Kohdansya, 1997.

[15] G. Nicolis and I. Prigogine, Exploring Complexity, An Introduction, San Francisco: Freeman and Company, 1989.

[16] G. Păun, Computing with membrane, J. Comput. Systems Sci., 61(1):108-143, Elsevier, 2001.

[17] Y. Suzuki, S. Tsumoto, and H. Tanaka, Analysis of Cycles in Symbolic Chemical System based on Abstract Rewriting System on Multisets. Proceedings of International Conference on Artificial Life V, pp. 482-489. MIT press, 1996.

[18] J.J. Tyson; P.C. Fife, Target patterns in a realistic model of the Belousov-Zhabotinskii reaction, J. Chem. Phys., 73, 2224-2237, 1980.

[19] J.J. Tyson, A quantitative amount of oscillation, bistability, and traveling waves in the Belousov-Zhabotinskii reaction, in: Oscillations and Traveling Waves in Chemical Systems, R.J. Field and M.Burger, eds. 93-144, Wiley, New York, 1985.

[20] M. Umeki and Y. Suzuki, A Simple Membrane Computing Method for Simulating Bio-Chemical Reactions, Computer and Informatics (in printing).

[21] M. Umeki and Y. Suzuki, On the simulation of the Oregonator by using Abstract Rewriting System on Multisets, 2nd Coupled Analysis Forum, Feb 2-3 2007, Hakata JAPAN, 2007.

Author index