Lecture Notes in Computer Science 5391

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

David Wolfe Corne Pierluigi Frisco
Gheorghe Păun Grzegorz Rozenberg
Arto Salomaa (Eds.)

Membrane Computing

9th International Workshop, WMC 2008
Edinburgh, UK, July 28-31, 2008
Revised Selected and Invited Papers

Springer

Volume Editors

David Wolfe Corne
Heriot-Watt University, School of Mathematical and Computer Science
Edinburgh, EH14 8AS, UK
E-mail: dwcorne@macs.hw.ac.uk

Pierluigi Frisco
Heriot-Watt University, School of Mathematical and Computer Science
Edinburgh, EH14 4AS, UK
E-mail: pier@macs.hw.ac.uk

Gheorghe Păun
Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania
E-mail: george.paun@imar.ro

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

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

Library of Congress Control Number: Applied for

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

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

ISSN 0302-9743
ISBN-10 3-540-95884-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-95884-0 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12605439 06/3180 5 4 3 2 1 0

Preface

This volume contains a selection of papers presented at the 9th Workshop on Membrane Computing, WMC9, which took place in Edinburgh, UK, during July 28–31, 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 Thessaloniki, Greece, in June 2007, with the proceedings published as volumes 2933, 3365, 3850, 4361, and 4860 of *Lecture Notes in Computer Science*.

The 2008 edition of WMC was organized at Heriot-Watt University, 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). This time most of the invited talks were given by speakers from outside the membrane computing community. The goal of this planing was to provide an opportunity for distinguished researchers from the bioinformatics and systems biology communities to learn about and to appreciate membrane systems as modeling platforms, while at the same time allowing the membrane computing community to learn about current challenges of bioinformatics and systems biology, especially those where membrane systems could be potentially useful.

The six invited speakers were: Vincent Danos (Edinburgh, UK), David Gilbert (Glasgow, UK), Paulien Hogeweg (Utrecht, The Netherlands), Markus Kirkilionis (Warwick, UK), Francisco-José Romero-Campero (Nottingham, UK), Stephen Wolfram (Champaign, IL - USA). Extended abstract or full papers related to most of these invited talks are included in the present volume.

All papers presented at the workshop were collected in a pre-proceedings volume, published by Heriot-Watt University; the volume was available during the workshop. Each of these papers was subject of at least three 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 (Seville, Spain), Oscar H. Ibarra (Santa Barbara, CA, USA), Fairouz Kamareddine (Edinburgh, UK), Lila Kari (London, Canada), Alica Kelemenová (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 (Milan, Italy).

A selection of 22 papers are included in the present volume. Most of them were significantly reworked after the meeting, taking also into account discussions that took place during the workshop.

Two prizes were awarded during WMC9: one for the *best paper*, and another for *important contributions to membrane computing*.

Three papers were nominated for the first prize:

1. P.A. Abdulla, G. Delzano, L. Van Begin: On the Qualitative Analysis of Conformon P Systems
2. M. Cardona, M.A. Colomer, M.J. Pérez-Jiménez, D. Sanuy, A. Margalida: Modeling Ecosystems Using P Systems: The Bearded Vulture, a Case Study
3. V. Nguyen, D. Kearney, G. Gioiosa: A Hardware Implementation of Nondeterministic Maximally Parallel Object Distribution in P Systems

The jury (consisting of D. Corne, P. Frisco, and Gh. Păun) chose the first paper for the award.

The prize for *important contributions to membrane computing* was awarded to Mario J. Pérez-Jiménez, from Seville University, Spain, "for his continuous and successful enrichment of this field through his exemplary work." The prize consisted of a one-year subscription to *Natural Computing* journal, published by Springer.

The Organizing Committee consisted of Pierluigi Frisco – Chair, David Corne – Co-chair, Elizabeth Bain Andrew – Secretary and David Li – Web page developer.

The most complete source of information about membrane computing is the P systems website from `http://ppage.psystems.eu`, and its mirror in China `http://bmc.hust.edu.cn/psystems`. The address of the workshop website (designed by David Li) is `http://macs.hw.ac.uk/wmc9`. The logo of the workshop was designed 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* published by Springer, Oxford University Press, and the Scottish Bioinformatics Forum.

October 2008

David Corne
Pierluigi Frisco
Gheorghe Păun
Grzegorz Rozenberg
Arto Salomaa

Table of Contents

Invited Lectures

Regular Papers

Investigation of a Biological Repair Scheme

Vincent Danos[1], Jérôme Féret[2], Walter Fontana[3],
Russell Harmer[4], and Jean Krivine[3]

[1] University of Edinburgh
[2] INRIA, CNRS, École Normale Supérieure
[3] Harvard Medical School
[4] CNRS, Université Paris-Diderot

Abstract. This note details an interaction pattern for the allocation of a scarce biological resource where and when it is needed. It is entirely based on a mass action stochastic dynamics. Domain-domain binding plays a crucial role in the design of the pattern which we therefore present using a rule-based approach where binding is an explicit primitive. We also a use a series of refinements, starting from a very simple interaction set, which we feel gives an interesting and intuitive rationale for the working of the final repair scheme.

1 Introduction

What is a scheme the reader will ask. As we understand it in this paper, a scheme or a pattern is a set of interactions between biological agents (we use the neutral term agent throughout this paper, but one can think of them as idealised proteins) which has a noteworthy property and which one can study and recognise under various guises in various systems. A well-known example is that of a reversible covalent modification system that can be made to behave, under well-chosen conditions, as an ultra-sensitive switch [1]. Another related one is that of an autophosphorylating kinase that shows bistable behaviour and can be used as a memory [2]. Importantly, the often non-intuitive kinetic and topological aspects of biological schemes are integral to their functioning. As such, they differ from static network motifs à la Alon [3] which are singled out because of their statistical properties, not their dynamical ones.

Consider a scheme where an agent $\mathtt{X}$ that can be activated by two upstream 'input' agents $\mathtt{I}_1$, $\mathtt{I}_2$, and can in turn activate two downstream 'output' ones $\mathtt{O}_1$, $\mathtt{O}_2$. Suppose that in the sole presence of $\mathtt{I}_1$ only $\mathtt{O}_1$ gets significantly activated. From these simple premisses one can obtain a wide range of behaviour. Specifically, $\mathtt{I}_2$ can be made to completely override the effect of $\mathtt{I}_1$, meaning that when $\mathtt{I}_2$ is present the only significant amount of downstream activity is that of $\mathtt{O}_2$ irrespective of the presence of $\mathtt{I}_1$. However this is only when the interaction rates, copy numbers (eg $\mathtt{X}$ must be saturated by the inputs $\mathtt{I}$s), and binding interfaces (eg $\mathtt{X}$ must use a different binding interface for the inputs $\mathtt{I}$s and for the outputs $\mathtt{O}$s) are well chosen. Thus the corresponding static motif would retain very little

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 1–12, 2009.

of the information on which the scheme hinges, it would be too abstract a view on it. Indeed many behaviours can be extracted even from the basic feed-forward motif [4] (see also Ref. [5] for an extensive discussion of the under-determination of motifs).

The situation is similar for the scheme we will define in this note. One has a target agent `T` that sporadically decays into a bad state. Repair is provided by an agent `K`, but in some rare cases also needs a helper agent `H`. The question is how to allocate this scarce and scarcely needed repair resource `H` so that repair proceeds efficiently. In a world of centralised computations, to get that resource when and where it is needed is a trivial question, as one can just order the `H`s to go where they are needed. Our scheme offers a distributed computational solution that relies entirely on mass action (ie on chance collisions). Incidentally, this pattern was taken form a larger model where it is combined with the setting up of a transient memory to obtain what one might call and error-correcting mechanism.[1] Indeed schemes will often be combined in a biological situation to obtain interesting effects. We are not addressing here the interesting and difficult question of how schemes should be composed, though.

There comes the question of what notation or language to use to represent and study such schemes. As said, they often use in a crucial fashion the dynamics of domain-domain binding -which means both the kinetic aspects of binding and the topological constraints induced by the agents' domain structures which may or may not allow simultaneous binding. There is a recent and growing recognition that such structural detail matters in the statistics of the static properties of protein networks [6]. We believe the same broadly holds for the understanding of their dynamics and have developed in the Kappa project [7] a simple notation for binding which is particularly useful for combinatorial situations (as eg in cellular signalling) and which we will use here. Note that Kappa is one of the few languages to propose such a direct notation (see also the BNG language [8]). Basic interactions are expressed as rules -not as reactions as in the traditional approach using differential equations or Petri nets, or as agent-centric interactions as in process-algebraic approaches [9].

As we will see below, the presentation of the scheme dynamics as a set of rules not only allows for a crisp description, but also allows one to derive the scheme by a succession of rule refinements, starting from a very basic model. We believe this derivation illuminates the function of our interaction pattern and demonstrates the relevance of refinement as an explanatory force (see Ref. [10] for more examples, and a rigourous account of refinement).

We postpone the discussion of the significance of the notion of refinement, and how one can think of it in relation to the actual evolution of protein networks, to the concluding remarks, and start with an informal presentation of our representational language Kappa (§2), and an outline of the paper (§3).

[1] A model of epigenetic information maintenance, unpublished work in collaboration with Arndt Benecke.

2 Kappa, Briefly

Kappa is a fully implemented stochastic calculus of binding and modification that addresses the modelling of fine-grained regulation in biological systems.[2] The unique construct known to Kappa is called an agent (usually meant to represent a protein), a name (eg identifying the corresponding protein) and a set of sites (eg the protein binding domains and modifiable amino-acids). Each site can carry a value, generally used to represent post-translational modifications. The set of values taken by the sites of an agent is commonly called the agent internal state, and an event that modifies the agent internal state is called a modification. Agents can also use sites to bind other agents -with the key restriction that a site can be used to bind at most one other site at a time. Often the modification of an agent results from another agent binding and modifying it. This is a natural way to represent, for instance, the basic interaction of a substrate with a kinase (as in the example right below).

Events such as binding, unbinding, and modification are described by rules specifying under which condition they may happen. This places Kappa in the family of rule-based modelling languages; another member of this small family is the BNG language. Rules have rates, and following the mass action principle, the likelihood that a given rule applies to a particular system x is proportional to the number of its instances in x multiplied by its rate, aka the rule's activity. The accompanying time advance is on average inverse to the cumulated activity of all rules operating on x. So Kappa is a quantitative framework. The particular case where agents have no sites, and therefore cannot be bound or modified is equivalent to stochastic Petri nets (aka flat chemical reactions, or multiset rewriting). In the presence of sites however, the underlying rewriting theory is richer and is best seen as a kind of stochastic graph rewriting (this point of view is developed in Ref. [10]).

Kappa uses concurrency concepts to dissect the intricate causation mechanisms one finds in protein networks [7]. Tracking down the fate of an agent and accessing the fine causal structure that produces an event of interest is useful as we will see below. In our particular example another interest of using Kappa is that we can derive it by a succession of refinements.

Before we turn to describing our strategy to derive our scheme, let us begin with a simple kinase and substrate example to explain the notation we use and illustrate the notion of refinement.

2.1 An Example

Consider a kinase `K`, and `T` its target, and suppose `T` has two phosphorylatable sites x and y; one can decompose each phosphorylation event in a triplet of elementary ones 1) `K` binds its target `T` at site x or y, 2) `K` may (but need not) phosphorylate the site to which it is bound, 3) `K` dissociates from `T`.

[2] The Kappa implementation includes a graphical interface, is free for academic usage, and can be obtained at `support@plectix.com`.

This translates in the following rules (three per sites in T):

$$\begin{aligned}
&\mathtt{K}(a), \mathtt{T}(x) \to \mathtt{K}(a^1), \mathtt{T}(x^1)\\
&r_1 := \mathtt{K}(a^1), \mathtt{T}(x^1) \to \mathtt{K}(a), \mathtt{T}(x)\\
&\mathtt{K}(a^1), \mathtt{T}(x^1_u) \to \mathtt{K}(a^1), \mathtt{T}(x^1_p)\\
&\mathtt{K}(a), \mathtt{T}(y) \to \mathtt{K}(a^1), \mathtt{T}(y^1)\\
&\mathtt{K}(a^1), \mathtt{T}(y^1) \to \mathtt{K}(a), \mathtt{T}(y)\\
&r_2 := \mathtt{K}(a^1), \mathtt{T}(y^1_u) \to \mathtt{K}(a^1), \mathtt{T}(y^1_p)
\end{aligned}$$

In the textual notation we are using here, internal states are shown as subscripts u (unphosphorylated) and p (phosphorylated) to the sites they are attached to; bonds are represented as shared superscripts across agents to indicate the two endpoints of a link. The number chosen to name a link is irrelevant, as long as it is unique. A double arrow indicates a reversible rule. The left hand side of a rule specifies a condition to trigger the rule, while the right hand side specifies the various changes occurring as a consequence of applying the rule. Thus the third reaction says that when K is bound to T at x, x may be phosphorylated.

Importantly, the sites of an agent need not all be present in a rule, eg in the first rule above, T does not mention y. Likewise, if a site is mentioned at all, its internal state may be left unspecified, eg in the same first rule one does not say whether x in T is phosphorylated or not. This is the trivial but crucial *'don't care, don't write'* convention: only the conditions bearing on the application of a rule need to be represented. Not to put to fine a point on it, the agility of the rule-based approach relies entirely on the ability to trigger events based on partial conditions, an ability that ordinary methods do not have, and process-algebraic methods only partially have (for a more extensive account of the rule-based accrued flexibility see Ref. [11]). Technically partial matches rely on a suitable notion of site graph morphism [10].

2.2 Variants

Another point worth noticing is that Kappa's fine-grained notation forces one to make one's mechanistic choices explicit. Let us consider a few possible variants (among many) of the rules above:

$$\begin{aligned}
&\mathtt{K}(a^1), \mathtt{T}(x^1, y_u) \to \mathtt{K}(a^1), \mathtt{T}(x^1, y_p)\\
&r'_1 := \mathtt{K}(a^1), \mathtt{T}(x^1_p) \to \mathtt{K}(a), \mathtt{T}(x_p)\\
&r'_2 := \mathtt{K}(a^1), \mathtt{T}(x^?_p, y^1_u) \to \mathtt{K}(a^1), \mathtt{T}(x^?_p, y^1_p)
\end{aligned}$$

The first rule extends the modification capacities of K by allowing it to modify T at site y while being bound at x (when y is free); one sometimes say K is processive in this case. The second rule r'_1 allows K to dissociate from T at x only when x is modified. The third rule r'_2 forces the modification order, that is to say y can only be modified after x is; it introduces a new notation, using a ? superscript on x to mean that x can be free or not -all that counts is that x is modified.

One can combine the above variants, and exchange the roles of x, and y to obtain a rather large set of variations on what is presumably the simplest scheme,

ie covalent modification. These various choices are just as key to the dynamics as rule rates are. As said in the introduction access to this fine-grained description is crucial to the definition of a scheme.

The variant rules r_1', r_2' have in common that they are more specific than respectively r_1, and r_2. For instance, r_1' now ensures that K does not let go of its substrate too early, whereas this was possible according to the original dissociation rule r_1. By setting a high rate r_1' one could also make sure it does let go of the substrate fast when it has been modified. (Presumably one will find cases where selection has led to the evolution of such smart enzymes.)

The substitution of r_1 with the smarter r_1', or r_2 with r_2' are simple examples of *refinement*, whereby a rule is replaced with one or more rules that are more specific than the initial rule. While working out in full generality when a refinement is neutral [10], ie when it preserves the underlying dynamics, is not easy, for the simple refinements we need here, an intuitive understanding of the notion will be enough.

3 Outline

Now that we have a better idea of the language we will use, let us explain how we organise the derivation and study of our pattern in the next section (§4).

We first consider a model where:

- a *target* agent $\mathtt{T_1}(k)$ has one site k which can be in one of two internal states, written $\mathtt{T_1}(k_{yes})$, and $\mathtt{T_1}(k_{no})$ (as in §2, internal states are shown as subscripts);
- damage happens to $\mathtt{T_1}$ which means its unique site can spontaneously switch or decay from the 'yes' to the 'no' state: $\mathtt{T_1}(k^?_{yes}) \rightarrow \mathtt{T_1}(k^?_{no})$
- damage can be subsequently repaired by an agent K.

(Remind that the superscript ? in the rule above indicates that this event may happen irrespective of whether k is bound or free.)

In order to calibrate the model and measure how well it is doing, we observe how many T_1s are free in the 'yes' state, and how many of these are 'sleeping' -meaning they are in the correct 'yes' state but still bound to K. It is rather easy then to build a basic model that works to our satisfaction in terms of efficiency, ie how fast $\mathtt{T_1}$ is repaired, and this concludes our first step.

The next step consists in perturbing the initial model by introducing a variant $\mathtt{T_2}$ of our initial target agent $\mathtt{T_1}$. This new agent $\mathtt{T_2}$ will have the same rules as $\mathtt{T_1}$ except for repair. Specifically, we will suppose that in order to repair $\mathtt{T_2}$, K needs to be bound to a helper agent H. Intuitively one can think of $\mathtt{T_2}$ as being more fragile or needing more work to be put back in the correct 'yes' state. Obviously, if no rule could sense the $\mathtt{T_1}$, $\mathtt{T_2}$ distinction the behaviour would be unchanged, but precisely the postulated fragility of $\mathtt{T_2}$ introduces such a distinction.

As we will see, this second step can presented by ways of refinement. Under quite general conditions it will break down the repair mechanism, the reason for this being that Hs are distributed uniformly across Ks and will have no reason to

be where they are needed, ie with these particular K$_s$ that happen to bind a T$_2$ and not a T$_1$. In fact, the smaller the fraction of T$_2$ and the more their necessary helpers H will be 'diluted'.

So the next and last step aims at repairing the repair system. One cannot order an H to go where it is needed, however it is entirely possible to trap it there and have it stay longer. To do so, it suffices to refine the rule in charge of the dissociation of the KH bond and modulate its rate depending on whether K is bound to a T$_2$ (low rate) or not (high rate, including the case K is not bound to a target). With this counter-refinement one obtains the pattern we are interested in. No central scheduling mechanism directs H$_s$ to T$_2$s, the refinement just makes the places in need of an H stickier. Yet this purely local trick works just as well as a centralised one would. It is an altogether different (and interesting) question to know whether this construct is also something one can engineer *in vivo*.

4 The Scheme

With our plan in place we can turn to the next section. All the code presented there is written in the syntax of the tool we use to examine the behaviour of our succession of models, and can be used as is. For this to be possible we switch now to a pure ascii notation, whereas so far we had used more mathematical notations with internal states as subscripts and bindings as exponents. Hopefully the slight change of syntax will feel natural.

4.1 A Simple Repair Model

To begin with we have three agents K(t,h), H(k), T$_1(k)$ with the interaction rules shown below. Stochastic rule rates are indicated on the right of each rule.[3]

```
K(t!1),T1(k!1) -> K(t),T1(k) @ 100
K(t),T1(k) -> K(t!1),T1(k!1) @ 1
K(t!1),T1(k~no!1) -> K(t!1),T1(k~yes!1) @ 50

T1(k~yes?) -> T1(k~no?) @ 1

K(h!1),H(k!1) -> K(h),H(k) @ 500
K(h),H(k) -> K(h!1),H(k!1) @ 10
```

One recognises the first three rules as a simple modification triplet, where neither the association, nor the dissociation rule are smart in the sense of the preceding discussion (§2.2). The fourth rule expresses T$_1$'s spontaneous decay. At this stage H plays no role, and is just a passive bystander that may bind K as specified in the last two rules. Not however, while the KH binding has the same equilibrium constant has that of the KT one, we have this binding more labile by using higher on- and off-rates. This will play a role in the implementation of our scheme.

[3] Remind that a stochastic bimolecular rate γ is related to its intensive deterministic analogue as $\gamma AV = k$ where V is a volume and A is Avogadro's number.

As initial conditions we pick:

```
%init: 10 * (K(h,t))
%init: 10 * (H(k))
%init: 100 * (T1(k~yes))
```

As for observables, meaning the objects that we want the simulation to keep track of, as said in §3, we pick the free $\mathtt{T_1}(k_{yes})$, the sleeping ones $\mathtt{T_1}(k^-_{yes})$ where $\mathtt{T_1}$ is bound on k and yet in a proper internal state, and $\mathtt{K}(h\text{-})$ the number of Ks bound to an H -an observable which allows us to monitor the saturation of K on its H side (of which more later):

```
%obs: T1(k~yes)
%obs: T1(k~yes!_)
%obs: K(h!_)
```

This completes the definition of our first model and with this choice of parameters, we find that at steady state about 60% of the $\mathtt{T_1}$s are repaired and free (Fig. 1). The remainder of the $\mathtt{T_1}$s can be considered as being under repair, including those 5% $\mathtt{T_1}$s that are already repaired but not yet released by K.

To ease the reading by reducing variance, copy numbers are rescaled by a factor of 10 in all plots (and bimolecular rates compensated accordingly).

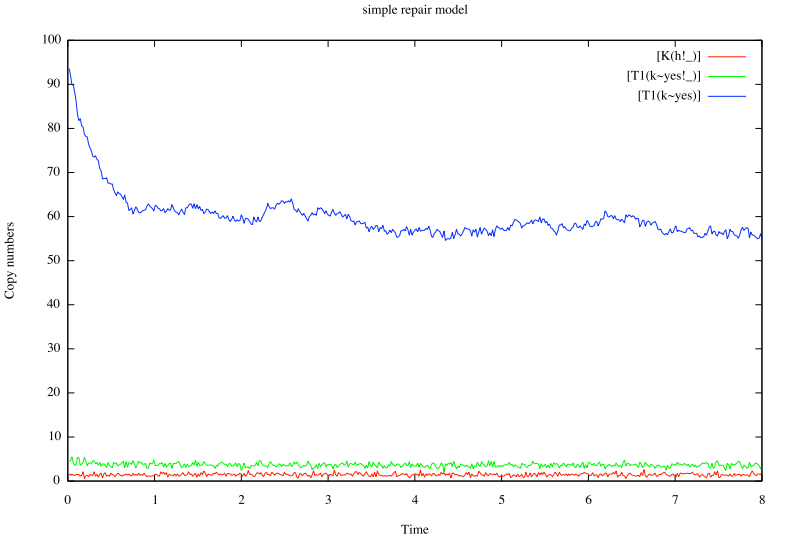


Fig. 1. A run of the simple repair model (rescale 10, 500 data points): about 60% of the $\mathtt{T_1}$s are free and in the 'yes' state; about another 5% are yet to be released; finally about 30% of the Ks are bound to an H

4.2 Introducing $\mathtt{T_2}$

We now take a strict duplicate of $\mathtt{T_1}$ which we call $\mathtt{T_2}$ with identical rules. To do this we could use a refinement introducing a fictitious site of $\mathtt{T_1}$ with an internal

state telling whether the agent is a T_1, or a T_2. (In general, one can encode agent names as internal states that no rule can ever change.)

Equivalently, we just copy the rules:

```
K(t!1),T2(k!1) -> K(t),T2(k) @ 100
K(t),T2(k) -> K(t!1),T2(k!1) @ 1
K(t!1),T2(k~no!1) -> K(t!1),T2(k~yes!1) @ 50

T2(k~yes?) -> T2(k~no?) @ 1
```

We also need to make room for our variant agent in the new initial state (`K`, `H` remain as before at 10 each). We choose here to divide evenly the population into 50% of T_1s and T_2s.

```
%init: 50 * (T1(k~yes))
%init: 50 * (T2(k~yes))
```

We add the following new observable

```
%obs: T2(k~yes)
```

As expected (Fig. 2) nothing changes and the evolution of T_2 is similar to that of T_1 modulo a rescale (inducing somewhat wider fluctuations).

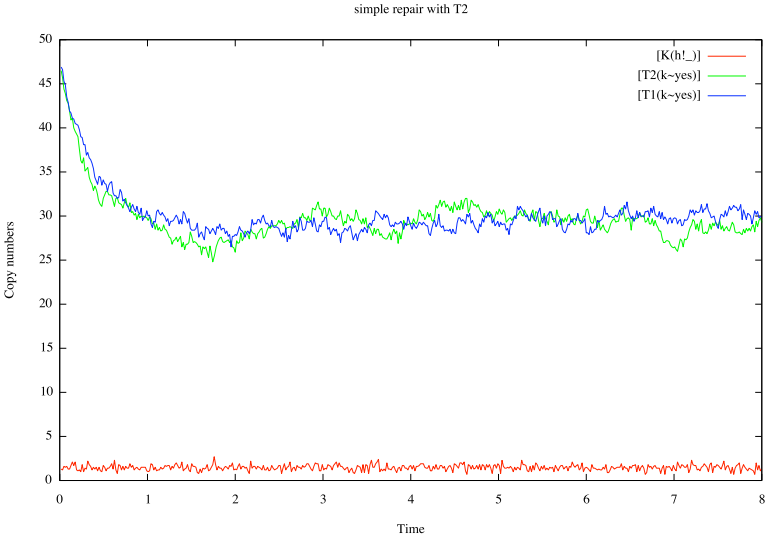


Fig. 2. The repair model with T_2 added (rescale 10, 500 data points): since no rule distinguishes T_1 and T_2: nearly 60% of the T_2s are free and repaired at steady state just as for the T_1s; one sees wider fluctuations because of the population divide

4.3 T_2's Specific H-Mediated Repair

Now is the time where we change our rule set significantly as we replace the above T_2 repair rule with the following:

```
K(t!1,h!_),T2(k~no!1) -> K(t!1,h!_),T2(k~yes!1) @ 50
```

whereby we specify that in order for K to be able to repair T_2, K must be bound to the helper agent H. This constitutes a (non neutral) refinement (the other case where K is free on h is assigned a zero rate). Note that in the actual formulation of the rule, it suffices to state that K is bound on its h site, since only H binds there. This convenient abbreviation would not work if other agents could bind K at h.

As a result we observe a sharp degradation of T_2's repair (Fig. 3).

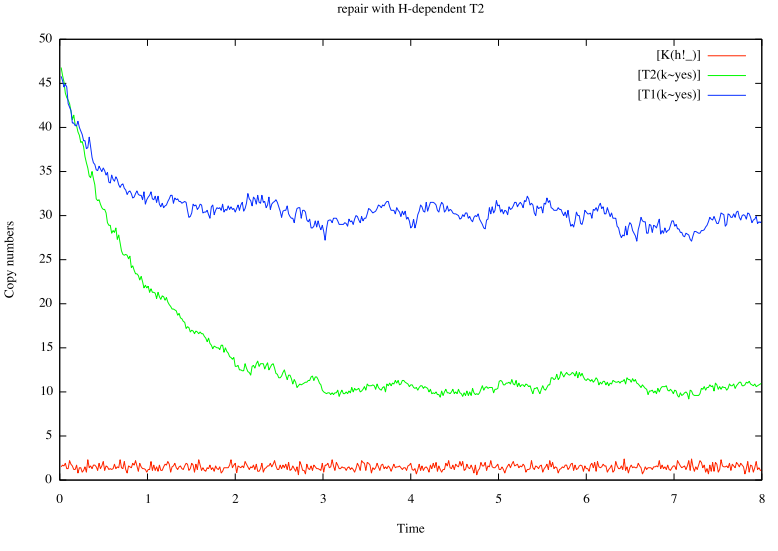


Fig. 3. The third repair model where T_2's repair is H-dependent (rescale 10, 500 data points): less than 20% of the T_2s are in the 'yes' state

Importantly, the behaviour obtained here depends on whether K is *saturated* by H, or in other words on how high the probability that a given K is bound to a T is. To monitor this probability we have added the number of bound Hs in each plot. This probability depends only on the equilibrium dissociation rate of the K and H binding and their initial copy numbers H and K. If it is 1, then T_1 and T_2 are again indistinguishable, since T_2 only behaves differently when H is absent, so T_2 will be repaired just as well as T_1. If it is 0, then T_2 will never be repaired. With our specific choices one has roughly 20% of the K bound to an H, so K is far from being saturated on its H binding site. Even in such intermediate conditions $T_2(k_{yes})$ goes down steeply -which will make the rescue (to come next) more spectacular. One might say that Hs do not find T_2s, and there is a 'stochastic dilution' effect.

4.4 Repairing T_2's Repair

To counter the stochastic dilution of the repair process all one needs is a refinement of the KH dissociation rule. Specifically, we replace that unconditional dissociation rule with the following three rules:

```
K(h!1,t),H(k!1) -> K(h,t),H(k) @ 1000
K(h!1,t!2),H(k!1),T1(k!2) -> K(h,t!2),H(k),T1(k!2) @ 1000
K(h!1,t!2),H(k!1),T2(k!2) -> K(h,t!2),H(k),T2(k!2) @ 0.001
```

We are trying to trap H when a T_2 stands on the other site of K. Thus H will reside longer where it is needed. In the other cases, that is when K is either free on t, or bound to a T_1, we increase the dissociation rate so that T_2s do not linger. The fact that we have chosen early on high on- and off-rates for the KH binding enhances the efficiency of this strategy, since H will explore quickly its potential partners until it finds one that is engaged in the repair of a T_2.

With the above refinement and rate manipulation, one obtains a far more efficient repair rate for T_2, as one can see on Fig. 4. It is important to note that this is not obtained by pushing the KH binding into saturation; indeed, as one can also see, the occupancy rate of Hs has not increased.

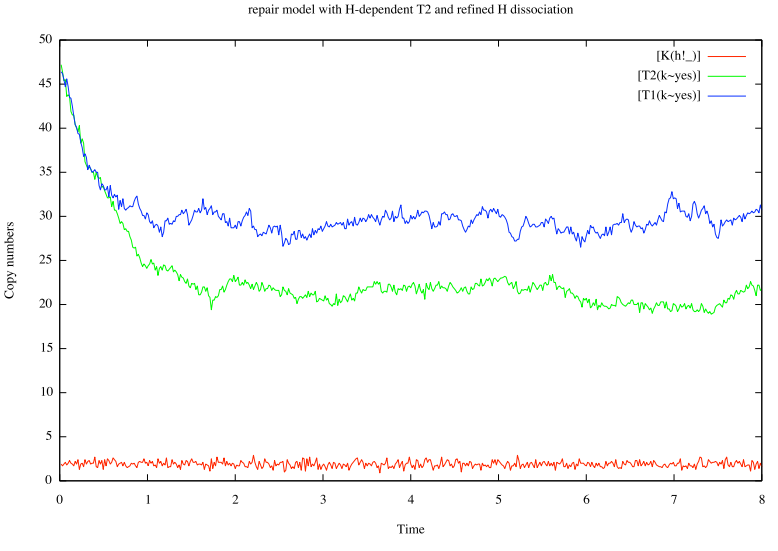


Fig. 4. The last repair model obtained by refining KH dissociation: nearly 50% of the T_2s are free and repaired, which is better than before (Fig. 3), and nearly as good as in the original model (Fig. 2). Take note that there is no significant increase in the H binding to K (lower red curve).

In a biological situation, presumably T_2 will have a function which it can only perform when in its correct 'yes' state, and the additional availability afforded by the refinements above could make all the difference.

5 Conclusion

In the repair scheme we have presented, one has a repair agent K which can recognise different targets, and bring a helper H to to participate in the process for some specific targets. The question is how to get the said helper where it is needed. An easy solution would be to saturate K, in a manner vaguely reminiscent of a Swiss knife, but this begs the question of whether there is perhaps a more elegant and parsimonious solution, where the knife would self-assemble only where and when needed. Indeed, there is one which we have shown in this note, and which one could compare to a stochastic ratchet. This pattern of interaction could extend further than the particular confines of the problem we meant to elucidate in this note -clearly.

With the careful derivation of this scheme, we have made a numerical case for our initially informal intuition, ie we have shown it made numerical sense. This is one of the traditional role of modelling in biology, to strengthen one's assumptions by putting them to numerical test. This we have done purely by simulations, and it would be very interesting to pursue this investigation with a more mathematically grounded approach. Perhaps with a set suitable simplifying assumptions one could have an entirely analytical approach of the problem, which would greatly help to understand how the several parameters at play (rates and copy numbers) impinge on the efficiency of our scheme. In particular, with a purely numerical construction as the one we offered here, it is unclear how the scheme behaves if one changes the fractions of the various repair targets.

As said in the introduction, this scheme was placed in combination with others in a simple model of an aspect of epigenetic repair. In this larger model the helper H stochastic dilution is even more of a problem since non repairing the target on time compromises a temporary memory and leads to permanent mistakes, not just inefficiencies. An interesting consequence of the above investigation is that we have an example where very labile binding (meaning with high off- and on-rates) of the repair agent K to its target is crucial. It has already been observed that key dissociation rates in epigenetic repair are higher than would seem reasonable. It may be that one will observe labile fast diffusing agents as well.

Finally, another point worth noticing is that the scheme was derived by a succession of refinements. In so doing we may also have given some substance to the idea that this scheme can be rather easily 'evolved' by variation and subsequent selection. This is a question that we wish to return to in the future.

References

1. Goldbeter, A., Koshland, D.: An Amplified Sensitivity Arising from Covalent Modification in Biological Systems. Proceedings of the National Academy of Sciences 78(11), 6840–6844 (1981)
2. Lisman, J.E.: A mechanism for memory storage insensitive to molecular turnover: a bistable autophosphorylating kinase. Proc. Natl. Acad. Sci. U S A 82(9), 3055–3057 (1985)

3. Alon, U.: Network motifs: theory and experimental approaches. Nat. Rev. Genet. 8(6), 450–461 (2007)
4. Wall, M.E., Dunlop, M.J., Hlavacek, W.S.: Multiple functions of a feed-forward-loop gene circuit. J. Mol. Biol. 349(3), 501–514 (2005)
5. Mazurie, A., Bottani, S., Vergassola, M.: An evolutionary and functional assessment of regulatory network motifs. Genome Biol. 6(4), R35 (2005)
6. Yeates, T.O., Beeby, M.: Proteins in a small world. Science 314(5807), 1882–1883 (2006)
7. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Rule-based modelling of cellular signalling. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 17–41. Springer, Heidelberg (2007)
8. Blinov, M.L., Faeder, J.R., Goldstein, B., Hlavacek, W.S.: A network model of early events in epidermal growth factor receptor signaling that accounts for combinatorial complexity. BioSystems 83, 136–151 (2006)
9. Priami, C., Regev, A., Shapiro, E., Silverman, W.: Application of a stochastic name-passing calculus to representation and simulation of molecular processes. Information Processing Letters (2001)
10. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Rule-based modelling, symmetries, refinements. In: Fisher, J. (ed.) FMSB 2008. LNCS (LNBI), vol. 5054, pp. 103–122. Springer, Heidelberg (2008)
11. Danos, V.: Agile Modelling of Cellular Signalling. Computation in Modern Science and Engineering 2, Part A 963, 611–614 (2007)

An Introduction to BioModel Engineering, Illustrated for Signal Transduction Pathways

David Gilbert[1], Rainer Breitling[2], Monika Heiner[3], and Robin Donaldson[4]

[1] School of Information Science, Computing and Mathematics, Brunel University, UK
david.gilbert@brunel.ac.uk
[2] Groningen Bioinformatics Centre, University of Groningen,
9751 NN Haren, The Netherlands
[3] Department of Computer Science, Brandenburg University of Technology,
Cottbus, Germany
[4] Bioinformatics Research Centre, University of Glasgow, Glasgow G12 8QQ, UK

Abstract. BioModel Engineering is the science of designing, constructing and analyzing computational models of biological systems. It is inspired by concepts from software engineering and computing science.

This paper illustrates a major theme in BioModel Engineering, namely that identifying a quantitative model of a dynamic system means building the structure, finding an initial state, and parameter fitting. In our approach, the structure is obtained by piecewise construction of models from modular parts, the initial state is obtained by analysis of the structure and parameter fitting comprises determining the rate parameters of the kinetic equations. We illustrate this with an example in the area of intracellular signalling pathways.

1 Introduction

BioModel Engineering takes place at the interface of computing science, mathematics, engineering and biology, and provides a systematic approach for designing, constructing and analyzing computational models of biological systems. Some of its central concepts are inspired by efficient software engineering strategies. BioModel Engineering does not aim at engineering biological systems per se, but rather aims at describing their structure and behavior, in particular at the level of intracellular molecular processes, using computational tools and techniques in a principled way.

In this paper we present some techniques for the systematic construction of models of biochemical systems from constituent building blocks, and how such models can be tuned to exhibit some desired behaviour, applying our approach to a signal transduction pathway. Our presentation is aimed at computing scientists and software engineers who want to learn how their skills can be successfully applied in modern systems biology.

After a brief introduction of the biological context, this paper illustrates a major theme in BioModel Engineering, namely that identifying a (qualitative) model means (1) finding the structure, (2) obtaining an initial state and (3)

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 13–28, 2009.

parameter fitting. In our approach, the structure is obtained by piecewise construction of models from modular parts, the initial state which describes concentrations of species or numbers of molecules is obtained by analysis of the structure and parameter fitting comprises determining the rate parameters of the kinetic equations by reference to trusted data.

2 Biochemical Context

There are many networks of interacting components known to exist as part of the machinery of living organisms. Biochemical networks can be metabolic, regulatory or signal transduction networks.

In this paper we focus on signal transduction, which is the mechanism which enables a cell to sense changes in its environment and to make appropriate responses. The basis of this mechanism is the conversion of one kind of signal into another. Extracellular signaling molecules are detected at the cell membrane by being bound to specific trans-membrane receptors that face outwards from the membrane and trigger intracellular events, which may eventually effect transcriptional activities in the nucleus. The eventual outcome is an alteration in cellular activity including changes in the gene expression profiles of the responding cells. These events, and the molecules that they involve, are referred to as (intracellular) "signalling pathways"; they contribute to the control of processes such as proliferation, cell growth, movement, apoptosis, and inter-cellular communication. Many signal transduction processes are "signalling cascades" which comprise a series of enzymatic reactions in which the product of one reaction acts as the catalytic enzyme for the next. The effect can be amplification of the original signal, although in some cases, for example the MAP kinase cascade, the signal gain is modest [22], suggesting that a main purpose is regulation [14] which may be achieved by positive and negative feedback loops [3], although there may be some feedback redundancy with respect to receptor internalisation [19].

3 Modelling Using Building Blocks Based on Enzymatic Reactions

Formally, a quantitative model of a biochemical pathway can be described by a tuple $< T, M, K, R >$ where T is the topology (biochemical species and their connectivities), M an initial state describing concentrations or molecular numbers of species, K a set of kinetic equations and R a set of kinetic rate parameters. A qualitative model comprises at least the topology T with optionally an initial state.

In this section we discuss how signal transduction cascades can be modelled in a modular fashion using a building-block approach, which will generate the topology T and set of kinetic equations K of a model. We have shown in previous work [4] how building-block based construction can be achieved using both a qualitative approach – Qualitative Petri nets, and quantitative approaches – Continuous Petri Nets and Ordinary Differential Equations (ODEs), but in this

review we restrict our discussions to ODEs. In the following sections we will introduce techniques to generate an initial state, or in Petri net terminology *marking* and obtain a set of rate parameters for the model.

The basic building block of any biological dynamic system is the enzymatic reaction: the conversion of a substrate into a product catalysed by an enzyme. Such enzymatic reactions can be used to describe metabolic conversions, the activation of signalling molecules and even transport reactions between various subcellular compartments. Enzymes greatly accelerate reactions in one direction (often by factors of at least 10^6), and most reactions in biological systems do not occur at perceptible rates in the absence of enzymes. We can illustrate a simple enzymatic reaction involving one substrate A, one product B, and an enzyme E by

$$A \xrightarrow{E} B \,. \tag{1}$$

3.1 Basic Kinetic Descriptions

In general, there are two ways to describe the kinetics of enzymatic reactions: *Michaelis-Menten* and *Mass-action*.

Michaelis-Menten
The *Michaelis-Menten* equation for the basic enzymatic reaction is given in Equation MM

$$V = V_{\max} \times \frac{[A]}{K_M + [A]} \tag{MM}$$

where V is the reaction velocity, $V_{\max}$ is the maximum reaction velocity, and K_M, the *Michaelis constant*, is the concentration of the substrate at which the reaction rate is half its maximum value. The concentration of the substrate A is represented by $[A]$ in this rate equation. With the total enzyme concentration $[E_T]$ and the equation

$$k_{\text{cat}} = \frac{V_{\max}}{[E_T]} \tag{2}$$

we are able to write the differential equations describing the consumption of the substrate and production of the product as:

$$\frac{d[A]}{dt} = -\frac{d[B]}{dt} = -kcat \times [E_T] \times \frac{[A]}{(K_M + [A])} \tag{3}$$

The Michaelis-Menten equation only holds at the initial stage of a reaction before the concentration of the product is appreciable, and makes the following assumptions:

1. The concentration of product is (close to) zero.
2. No product reverts to the initial substrate.
3. The concentration of the enzyme is much less than the concentration of the substrate, i.e. $[E] \ll [A]$.

Although these are reasonable assumptions for enzyme assays in a test tube, assumptions 1 and 2 do not hold for most metabolic pathways *in vivo*, and none of the assumptions is correct for cellular signalling pathways.

Mass-Action

A more detailed description using Mass-action kinetics can be given by taking into account the mechanism by which the enzyme acts, namely by forming a complex with the substrate, modifying the substrate to form the product, and a disassociation occurring to release the product, i.e. $A + E \rightleftharpoons AE \rightarrow B + E$. In order to take into account the kinetic properties of many enzymes, we associate *rate constants* with each reaction. Thus the enzyme E can combine with the substrate A to form the $A|E$ complex with rate constant k_1. The $A|E$ complex can dissociate to E and A with rate constant k_2, or form the product B with rate k_3:

$$A + E \underset{k_2}{\overset{k_1}{\rightleftarrows}} A|E \xrightarrow{k_3} B + E \ . \tag{MA1}$$

This simple mass-action model is related to the Michaelis-Menten Equation MM as described previously by the following constraints:

$$\frac{k_2 + k_3}{k_1} = K_M \tag{4}$$

where $k_3 = k_{\text{cat}} = \frac{V\text{max}}{[E_T]}$ as in Equation 2.

We can derive a set of differential equations, see Equation 5 from the mass-action description given in Equation MA1:

$$\begin{aligned}
\tfrac{d[A]}{dt} &= -k_1 \times [A] \times [E] + k_2 \times [A|E] \\
\tfrac{d[A|E]}{dt} &= \ \ k_1 \times [A] \times [E] - k_2 \times [A|E] - k_3 \times [A|E] \\
\tfrac{d[B]}{dt} &= \qquad\qquad\qquad\qquad + k_3 \times [A|E] \\
\tfrac{d[E]}{dt} &= -k_1 \times [A] \times [E] + k_2 \times [A|E] + k_3 \times [A|E]
\end{aligned} \tag{5}$$

The mass-action model described in Equation MA1 assumes that almost none of the product reverts back to the original substrate, a condition that holds at the initial stage of a reaction before the concentration of the product is appreciable. This means that this type of mass-action model is a direct equivalent of the Michaelis-Menten equation, and will face the same limitations when applied to *in vivo* signalling systems. However, the mass-action description offers much more flexibility and thus can be easily expanded to cover more general situations, for example Equations MA2 and MA3 below.

$$A + E \underset{k_2}{\overset{k_1}{\rightleftarrows}} A|E \xrightarrow{k_3'} B|E \underset{k_1'}{\overset{k_2'}{\rightleftarrows}} B + E \tag{MA2}$$

$$A + E \underset{k_2}{\overset{k_1}{\rightleftarrows}} A|E \underset{k_4'}{\overset{k_3'}{\rightleftarrows}} B|E \underset{k_1'}{\overset{k_2'}{\rightleftarrows}} B + E \tag{MA3}$$

3.2 Modelling One Step in a Signal Transduction Cascade

One step in a classical signal transduction cascade comprises the *phosphorylation* of a protein by an enzyme S which is termed a kinase, see Figure 1. It is the phosphorylated form R_p which can act as an enzyme to catalyse the phosphorylation of a further component in the cascades, see Figure 3(a).

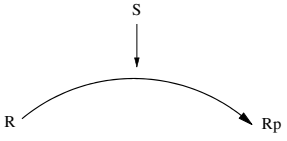


Fig. 1. Basic enzymatic step; R – signalling protein; Rp – phosphorylated form; S – kinase

We can model this reaction using any of the kinetic patterns introduced in Section 3.1 e.g., the Mass-action **MA1** pattern as follows, straightforwardly adapted from Equation MA1 by renaming in Equation 6, where R is a protein and R_p its phosphorylated form, S is a signal enzyme and $R|S$ the complex formed from R and S:

$$R + S \underset{k_2}{\overset{k_1}{\rightleftarrows}} R|S \xrightarrow{k_3} R_p + S \tag{6}$$

In order to ensure that such a single step is not a 'one shot' affair (i.e. to ensure that the substrate in the non-phosphorylated form is replenished and not exhausted), and hence that the signal can be deactivated where necessary, biological systems employ a phosphatase which is an enzyme promoting the dephosphorylation of a phosphorylated protein. This is depicted in Figure 2, which we are going to model by our Mass-action pattern.

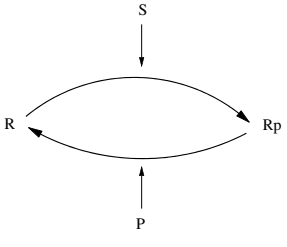


Fig. 2. Basic phosphorylation–dephosphorylation step; R – signalling protein; Rp – phosphorylated form; S – kinase; P – phosphatase

Using the **MA1** pattern (Mass-action kinetics) we get Equation 7, where P is a phosphatase and k_n, kr_n are rate constants for the forward and reverse reactions respectively. In many cases it would also be justified to model the dephosphorylation as an un-catalysed first-order decay reaction, because detailed

knowledge of phosphatase concentrations, specificities, and kinetic parameters is still lagging behind our understanding of the kinase enzymes.

$$\begin{aligned} R + S &\underset{k_2}{\overset{k_1}{\rightleftarrows}} R|S \xrightarrow{k_3} R_p + S \\ R + P &\xleftarrow{kr_3} R|P \underset{kr_2}{\overset{kr_1}{\rightleftarrows}} R_p + P \end{aligned} \tag{7}$$

3.3 Composing Kinase Cascades Using Building Blocks

Once we have defined the building blocks, we can compose them by chaining together basic phosphorylation–dephosphorylation steps.

Vertical and Horizontal Composition. Composition can be performed vertically as in Figure 3(a) to form a *signalling cascade*, where the signalling protein in the second stage is labelled RR and its phosphorylated form is labelled RR_p. Horizontal composition is illustrated in Figure 3(b) where a double phosphorylation step is described; the double phosphorylated form of a protein is subscripted by pp.

We can again use any of the kinetic patterns that were previously introduced in order to derive the models. For example, using **MA1** we can represent a two-stage cascade illustrated in Figure 3(a) by the following mass-action equations, ignoring for the sake of simplicity the dephosphorylation steps in the textual representation. The rate constants associated with the second stage are labelled kk_n. We would not expect the dephosphorylation rate constants to be related to the phosphorylation rate constants.

$$\begin{aligned} R + S_1 &\underset{k_2}{\overset{k_1}{\rightleftarrows}} R|S_1 \xrightarrow{k_3} R_p + S1 \\ RR + R_p &\underset{kk_2}{\overset{kk_1}{\rightleftarrows}} RR|R_p \xrightarrow{kk_3} RR_p + R_p \end{aligned} \tag{8}$$

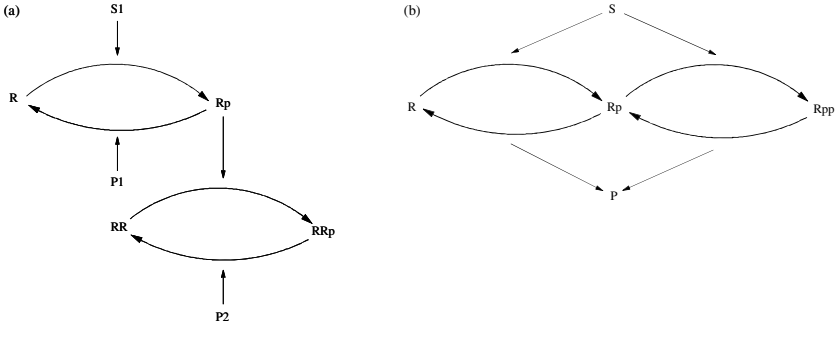


Fig. 3. (a) Vertical composition: Cascade formed by chaining two basic phosphorylation-dephosphorylation steps. (b) Horizontal composition: One stage cascade with a single to double phosphorylation step.

The addition of a double phosphorylation step to a cascade layer is given in Figure 3(b), where both the single and double phosphorylation steps are catalysed by the same enzyme S; likewise, the two dephosphorylation steps are usually catalysed by the same phosphatase P. This system component can be described by Equation 9, if we apply the mass-action kinetics **MA1** and ignore again for the sake of simplicity the dephosphorylation steps in the textual representation. The rate constants associated with the double phosphorylation are labelled kp_n. Often, we can assume that the rate constants for the two steps of the double phosphorylation are similar to those for the single phosphorylation.

$$\begin{aligned} R + S &\underset{k_2}{\overset{k_1}{\rightleftarrows}} R|S \xrightarrow{k_3} R_p + S \\ R_p + S &\underset{kp_2}{\overset{kp_1}{\rightleftarrows}} R_p|S \xrightarrow{kp_3} R_{pp} + S \end{aligned} \tag{9}$$

3.4 Negative and Positive Feedback

Feedback in a signalling network can be achieved in several ways. For example, negative feedback can be implemented at the molecular level by sequestration of the input signal S_1 by the product of the second stage RR_p. This system is sketched in Figure 4(a), and can be achieved by combining equations 8 and 10.

Similarly, positive feedback can also be achieved by the sequestration of the input signal S_1 by the product of the second stage, under the additional condition that the resulting $S_1|RR_p$ complex is a more active enzyme than S_1 alone. In this case we add Equation 11 to equations 8 and 10. The system is sketched in Figure 4(b).

$$S_1 + RR_p \underset{i_2}{\overset{i_1}{\rightleftarrows}} S_1|RR_p \tag{10}$$

$$R + S_1|RR_p \underset{kp_2}{\overset{kp_1}{\rightleftarrows}} R|S_1|RR_p \xrightarrow{kp_3} R_p + S_1|RR_p \tag{11}$$

Many other molecular mechanisms can be envisaged and are in fact observed in biological systems. All of these can be represented using the same basic formalism. For example, we can model an influence of RR_p on the phosphatase P_1, in which case the effects of positive and negative feedback are reversed, i.e. sequestration of P_1 by RR_p can cause positive feedback – see Figure 4(c). This can be achieved with Equations 8 and 12. Alternatively the situation where the $P_1|RR_p$ complex is more active than P_1 will cause negative feedback, Figure 4(d), and can be described by adding Equation 13 to Equations 8 and 12.

$$P_1 + RR_p \underset{i_2}{\overset{i_1}{\rightleftarrows}} P_1|RR_p \tag{12}$$

$$R_p + P_1|RR_p \underset{kr'_2}{\overset{kr'_1}{\rightleftarrows}} R_p|P_1|RR_p \xrightarrow{kr'_3} R + P_1|RR_p \tag{13}$$

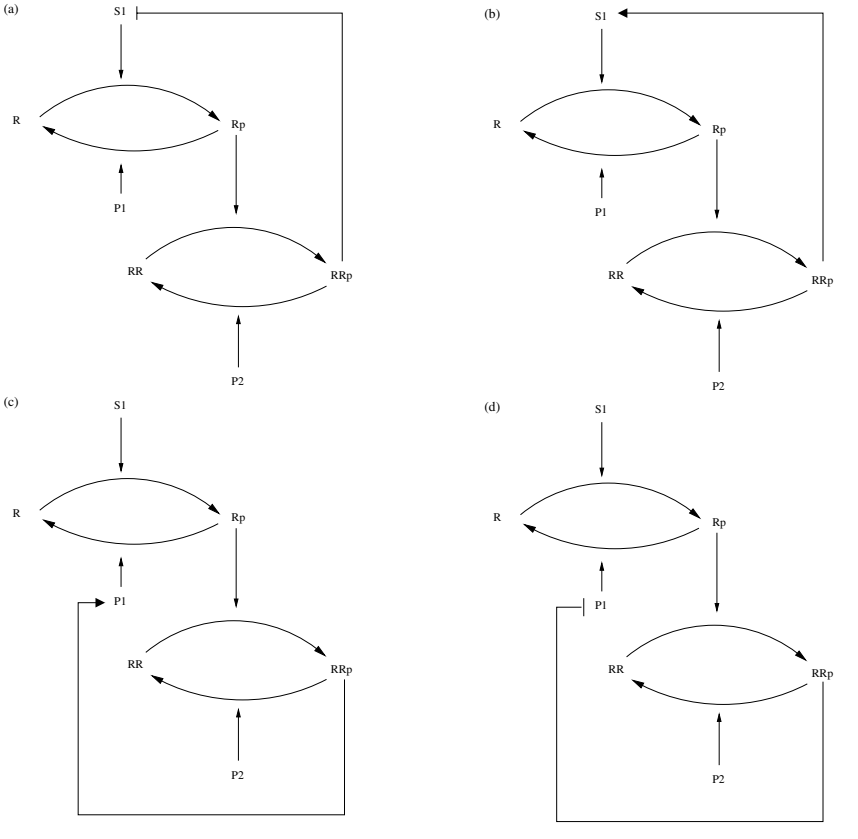


Fig. 4. Two-stage cascade with (a) negative feedback, (b) positive feedback; alternative two-stage cascade with (c) negative feedback, (d) positive feedback

4 Deriving Initial States from Qualitative Descriptions

Recall that a quantitative model of a biochemical pathway can be described by a tuple comprising the topology, an initial marking or set of concentrations, a set of kinetic equations and a set of rate parameters. Using the approach described in the previous section we have shown how to construct an initial model with a topology and set of kinetic equations. In this section we outline an approach to generate the initial steady state, and in the following section we will describe one approach to obtain a set of rate parameters.

The most abstract representation of a biochemical network is *qualitative* and is minimally described by its topology, usually as a bipartite directed graph with

nodes representing biochemical entities or reactions, or in Petri net terminology *places* and *transitions*. Petri nets are a well-established technique for representing biochemical networks, outlined e.g. in [17], [12] and for a general introduction to Petri nets see [18].

The qualitative description can be further enhanced by the abstract representation of discrete quantities of species, achieved in Petri nets by the use of tokens at places. These can represent the number of molecules, or the level of concentration, of a species, and a particular arrangement of tokens over a network is called a *marking*.

A P-invariant stands for a set of places, over which the weighted sum of tokens is constant, independently of any firing. So, P-invariants represent token-preserving sets of places. In the context of metabolic networks, P-invariants reflect substrate conservations, while in signal transduction networks P-invariants often correspond to the several states of a given species (protein or protein complex). A place belonging to a P-invariant is obviously bounded.

A T-invariant has two interpretations in the given biochemical context. The entries of a T-invariant represent a multiset of transitions which by their partially ordered firing reproduce a given marking, i.e. they occur basically one after the other. The partial order sequence of the firing events of the T-invariant's transitions may contribute to a deeper understanding of the net behaviour.

The entries of a T-invariant may also be read as the relative transition firing rates of transitions, all of them occurring permanently and concurrently. This activity level corresponds to the steady state behaviour [21]. Independently of the chosen interpretation, the net representation of minimal T-invariants (the T-invariant's transitions plus their pre- and post-places and all arcs in between) characterize typically minimal self-contained subnetworks with an enclosed biological meaning.

In previous work [10] we have shown how to systematically generate initial markings from (unmarked) qualitative Petri net descriptions, which can then be used in corresponding quantitative models. This work took as a concrete example the RKIP pathway [7], which is a subset of the ERK signalling pathway.

Our approach was to systematically construct suitable initial states by P-invariants and then to check their suitability by T-invariants, which have to be feasible. In more detail, having initially created an unmarked place/transition Petri net, a systematic construction of the initial marking can be made by placing tokens on places, taking into consideration the following criteria:

- Each P-invariant needs at least one token.
- All (non-trivial) T-invariants should be feasible, meaning, the transitions, making up the T-invariant's multi-set can be fired in an appropriate order.
- Additionally, it is common sense to look for a minimal marking (as few tokens as possible), which guarantees the required behaviour.
- Within a P-invariant, choose the species with the most *inactive* (e.g. non-phosphorylated) or the *monomeric* (i.e. non-complexed) state.

In a previous paper [10] we created a discrete Petri net model of the RKIP pathway, and analysed the model to show that it enjoys several nice properties, among them boundedness, liveness, and reversibility. Moreover, the net is

covered by P-invariants and T-invariants, all of them having sensible biological interpretation, and it fulfills several special functional properties, which have been expressed in temporal logic. Using reachability graph analysis we identified 13 strongly connected states out of 2048 theoretically possible ones, which permit self-reinitialization of the Petri net. From the viewpoint of the discrete model, all these 13 states are equivalent and could be taken as an initial state resulting in exactly the same total (discrete) system behaviour. We then transformed the discrete Petri net into a continuous model and demonstrated empirically that in the ODE model the 13 initial states, derived from the validated discrete model, result in the same (continuous) steady state. Moreover, none of the other 2035 possible initial states result in a steady state close to that derived using those identified by reachability graph analysis. This approach for steady state analysis was also successfully applied in another case study [11] to a larger model of the core MAPK pathway created by Levchenko et al.[15].

5 Parameter Estimation Using Model Checking

Model checking is an automated technique for the analysis of reactive systems to check whether properties, often expressed as formulas of temporal logic, hold for a model of the system. This technique was originally developed to check models of technical systems [16], and has been more recently applied to biochemical networks e.g. [5]. In previous work [8] we have shown in detail how model checking based on temporal logic descriptions of behaviour can be used in parameter estimation; in this section we summarise the main results.

Temporal logic is well-suited to formally represent semi-quantitative descriptions given by biologists who are often unsure about exact values of biochemical species over time due to the nature of the wet-lab experimental technology, and will describe behaviour in a semi-quantitative manner. For example, "the concentration of the protein peaks within 2 to 5 minutes and then falls to less than 50% of the peak value within 60 minutes". A significant challenge is how to automatically build a model which conforms to semi-quantitative behaviour.

5.1 PLTLc

Linear-time Temporal Logic (LTL) [20] is the fragment of full Computational Tree Logic (CTL*) [6] without path quantifiers, implicitly quantifying universally over all paths. LTL has been introduced in a probabilistic setting in [2], and extended by numerical constraints over real value variables in [9]. PLTLc combines both extensions, complemented by the filter construct as used in Probabilistic Computational Tree Logic (PCTL) [13] and Continuous Stochastic Logic (CSL) [1]. We start with the LTL with numerical constraints (LTLc) syntax:

$$\begin{aligned}\phi \ ::= \ & X\phi \mid G\phi \mid F\phi \mid \phi U\phi \mid \phi R\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \neg\phi \mid \phi \rightarrow \phi \mid \\ & value = value \mid value \neq value \mid value > value \mid value \geq value \mid \\ & value < value \mid value \leq value \mid true \mid false\end{aligned}$$

Numerical constraints over free variables are defined in this logic through the inclusion of free variables denoted by $\$fVariable$ in the definition of $value$.

Regular variables are read-only values which describe the behaviour of the model, whereas free variables are instantiated during the model checking process to the range of values for which the temporal logic property holds. Free variables are defined to have integer domains initialised to $[0 \rightarrow \infty)$ and describe protein concentrations, numbers of molecules and time. Constraints over free variables, which involve equality/inequality and relational operators, restrict the domain of the free variable.

PLTLc enhances LTLc by the inclusion of a probability operator and filter construct, and the probabilistic interpretation of the domains for the free variables. The top-level definition of PLTLc is:

$$\psi \ ::= \ \mathbf{P}_{\trianglelefteq x}[\,\phi\,] \mid \mathbf{P}_{\trianglelefteq x}[\,\phi\{SP\}\,]$$

where ϕ is an LTLc expression. SP is a State Proposition defined to be ϕ without any temporal operator (X, G, F, U, R), and containing *no* free variables without a loss of expressivity. Note that the square and curly brackets are part of PLTLc. Given that $\trianglelefteq \in \{>, \geq, <, \leq\}$, $P_{\trianglelefteq x}$ is any inequality comparison of the probability of the property holding true, for example $P_{\geq 0.5}$. We also permit the expression $P_{=?}$ returning the value of the probability of the property holding true. We disallow equality testing of the probability, $P_{=x}$ because of the representation of real values and the semantics of their equality.

The semantics of PLTLc is defined over a finite set of finite paths through the system's state space – stochastic or deterministic simulations, or time series data recorded in wet lab experiments. Let a path π be a finite sequence of states describing the behaviour of a biochemical system, $\pi = s_0, s_1, ..., s_n$ $(n < \infty)$ and π^i be the subsequence of π starting from state s_i, $i \leq n$, thus $\pi^i = s_i, s_{i+1}, ..., s_n$. Each path in the set of paths can be evaluated to a boolean value as to whether ϕ or $\phi\{SP\}$ holds. When all paths are evaluated, the number of true values in the set over the size of the set yields the overall probability of the PLTLc property. Hence for a stochastic model, where the set of paths is typically > 1, the probability is in the range $[0 \rightarrow 1]$ and calculated through Monte Carlo approximation, whereas a continuous model which contains a single path has a probability of either 0 or 1.

5.2 Distance Metrics

The distance between a model's behaviour M and the desired behaviour M_{des} with respect to some property ψ can be calculated using a distance metric.

Perhaps the simplest definition of the metric is the square difference between the model's probability of exhibiting some behavioural property ψ, $P(\psi)$ and desired probability $P_{des}(\psi)$:

$$d_\psi(M, M_{des}) = |P(\psi) - P_{des}(\psi)|^2$$

This approach works well in the stochastic world where the model exhibits many behaviours and the probability of the property is in the range $[0 \rightarrow 1]$. However, in the continuous world there is a single behaviour and the probability is either 0 or 1, thus the metric is too coarse grained to be used in a search

algorithm in the continuous world. To be useful in the search algorithm, the distance metric should return a value which indicates whether altering the current model has caused its behaviour to be closer to the desired behaviour, therefore providing a gradient for the search algorithm to ascend. We have defined such a distance metric for continuous models using a residual sum of squares function over probabilistic domains of free variables - for more details see [8].

5.3 Computational System

We implemented a computational system called the Monte Carlo Model Checker with a Genetic Algorithm, MC2(GA). The purpose of this computational system is to estimate the parameters of a model to make it exhibit desired behavioural properties. A genetic algorithm is used to move models through parameter space to minimise their distance to the desired behaviour, checked using a model checker.

Each model in our MC2(GA) system has a fixed structure and is represented by a chromosome, which is a set of kinetic rate constant values to be estimated (the model's genes) within predefined ranges. The chromosome could equally include initial concentrations/masses.

In the initial generation, a population of models is created by assigning to each model random values within the ranges for the kinetic rate constants. Each model in the population is evaluated to a fitness value related to the distance of its behaviour to the desired behaviour, hence a model with a smaller distance to the desired behaviour has a higher fitness. Our approach is to vary models' kinetic rate constant values in order to maximise their fitness values.

5.4 Case Study: MAPK Pathway

The EGF signal transduction pathway conveys Epidermal Growth Factor signals from the cell membrane to the nucleus via the MAP Kinase cascade [14]. The core MAPK cascade can be stimulated by both Epidermal Growth Factor (EGF) as well as Nerve Growth Factor (NGF). The reaction of the cell to EGF stimulation is cell proliferation, however the response to NGF is cell differentiation. The EGF signal transduction pathway produces transient Ras, MEK and ERK activation whereas NGF stimulation produces sustained activation. The underlying differences of the models describing EGF and NGF stimulation is of key interest to biochemists.

Work reported in [3], which attempted to discover the quantitative differences in initial concentrations and kinetic rate constants between models of these pathways with fixed topology. The authors in that original paper varied the initial concentrations and kinetic rate constants within biochemically sensible ranges. Simulation was performed with the model using each parameter value in the range and the output was manually inspected for sustained Ras, MEK and ERK activation. A result of this work was the finding that a 40-fold increase in the kinetic rate constant of SOS dephosphorylation can change the behaviour of the model from transient activation to sustained activation. In our approach,

reported in [8] we showed that this analysis could be improved by constructing a formal definition of the desired behaviour in temporal logic, and using model checking of the desired behaviour to replace the manual inspection of the simulation outputs.

Characterising the Desired Pathway Behaviour. The behaviour of sustained Ras, MEK and ERK activation arising from NGF stimulation observed in wet-lab experiment was described in rather informal statements in the original paper [3].

"The level of RasGTP rapidly reaches a maximum of up to 20% of total Ras within 2 min [then] the level of RasGTP is sustained at around 8% of total Ras."

Similar statements were made about sustained MEK and ERK activation. We formalised these statements using semi-quantitative PLTLc such that a model could be automatically checked for these behaviours using the MC2(PLTLc) model checker. We formalised these statements in a way to account for biological error by relaxing the constraints, for example that the stable level of RasGTP is 8% to between 5% and 10%:

Sustained Ras: Active Ras peaks within 2 minutes to a maximum of 20% of total Ras and is stable between 5% and 10% from at least 15 minutes

$$\mathbf{P}_{=?}\ [\ (\ d(\text{active Ras})\ >\ 0\ )\ \wedge\ (\ d(\text{active Ras})\ >\ 0\ )\ U\ (\ \text{time}\ \leq\ 2\ \wedge\ \text{active Ras}\ \geq\ 0.15{*}\text{total Ras}\ \wedge\ \text{active Ras}\ \leq\ 0.2{*}\text{total Ras}\ \wedge\ d(\text{active Ras})\ <\ 0\ \wedge\ (\ d(\text{active Ras})\ <\ 0\ \wedge\ \text{time}\ <\ 15\ )\ U\ (\ G(\ (\text{active Ras})\ \geq\ 0.05{*}\text{total Ras}\ \wedge\ \text{active Ras}\ \leq\ 0.10{*}\text{total Ras}\ )\ )\ )\ ]$$

where the protein RasGTP is found in isolation and in two complexes, thus active Ras $= RasGTP + Ras\_Raf + Ras\_GAP$ and total Ras $= RasGTP + Ras\_Raf + Ras\_GAP + RasGDP + Ras\_ShcGS$.

Genetic Algorithm. We first implemented a fitness function for use in MC2 (GA) to describe how close a model is to sustained activation. The descriptions of sustained Ras, MEK and ERK activation given earlier were not particularly helpful in the continuous setting due to the probability being simply 0 or 1. A fitness function based on a description which includes free variables allows greater expressivity using the probabilistic domains. Hence, we have rewritten these descriptions of sustained behaviours using free variables, for example:

Sustained Ras with free variables: Active Ras peaks within 2 minutes to a maximum of 20% of total Ras and is stable between any value in $\$RasTail1$ and any value in $\$RasTail2$ from at least 15 minutes

$$\mathbf{P}_{=?}\ [\ (\ d(\text{active Ras})\ >\ 0\ )\ \wedge\ (\ d(\text{active Ras})\ >\ 0\ )\ U\ (\ \text{time}\ \leq\ 2\ \wedge\ \text{active Ras}\ \geq\ 0.15{*}\text{total Ras}\ \wedge\ \text{active Ras}\ \leq\ 0.2{*}\text{total Ras}\ \wedge\ d(\text{active Ras})\ <\ 0\ \wedge\ (\ d(\text{active Ras})\ <\ 0\ \wedge\ \text{time}\ <\ 15\ )\ U\ (\ G(\ \text{active Ras}\ \geq\ \$RasTail1\ \wedge\ \text{active Ras}\ \leq\ \$RasTail2\ )\ )\ )\ ]$$

We then applied our computational system to find novel parameter sets which exhibit the desired behaviour. We estimated the values of a set of 16 critical

parameters identified as being potential candidates for modification by individually scanning all parameters and model checking the resulting simulation outputs. We also applied MC2(GA) to the critical parameters without V_28, to assess whether V_28 is crucial to achieving sustained activation. We found that if the critical parameters are estimated with V_28, then the convergence is quicker and the best model returned was fitter. The best model returned when estimating the critical parameters had fitness value 1, whereas with V_28 removed the best model returned had a fitness value approximately 0.93.

Figure 5 shows the output of one of the best model returned when estimating the critical parameters with and without V_28. Both behaviours showed good similarity (visually and in terms of fitness value) to the behaviour of the NGF signalling pathway outlined in the original paper. We also found that we can achieve a model with fitness value 1 through a 16-fold increase of V_28, compared with the original paper's 40-fold increase, if we also vary the other critical parameters.

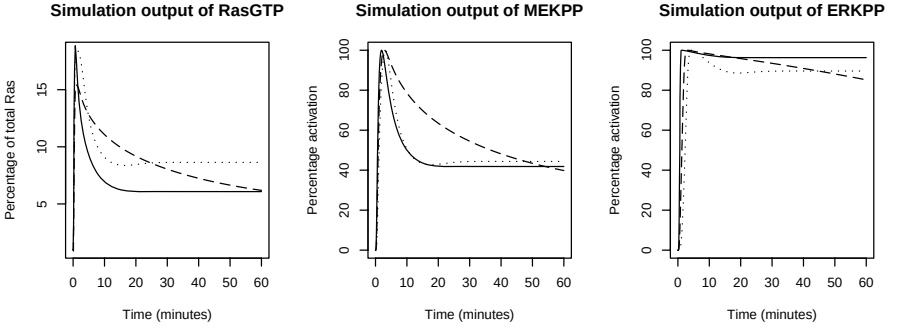


Fig. 5. The original model of the NGF signalling pathway (dotted) compared with the best model returned when varying the critical parameters (solid) and when varying the critical parameters without V_28 (dashed). The best model returned when varying the critical parameters only required a 16-fold increase in V_28 to achieve fitness value 1.

6 Conclusions

In this paper we have introduced the area of BioModel Engineering which is the science of designing, constructing and analyzing computational models of biological systems. We have illustrated some of the essential activities which BioModel Engineering encompasses - namely the construction of models of biological systems within a rigorous design framework, and techniques for the identification of initial start states, and the determination of rate parameters. We have presented these concepts in a practical manner, by way of an example in the area of intracellular signalling pathways. Our method is based on a modular building-block approach to the construction of network topology and associated biochemical equations, combined with analytical techniques from Petri nets for the determination of suitable start-states, and a novel model checking approach to drive the fitting of kinetic parameters.

Acknowledgments

DR was supported for his work on model checking by the SIMAP project which is funded by the European Commission framework 6 STREP programme.

References

1. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.K.: Verifying Continuous-Time Markov Chains. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 269–276. Springer, Heidelberg (1996)
2. Baier, C.: On Algorithmic Verification Methods for Probabilistic Systems. Habilitation thesis, University of Mannheim (1998)
3. Brightman, F.A., Fell, D.A.: Differential feedback regulation of the mapk cascade underlies the quantitative differences in egf and ngf signalling in pc12 cells. FEBS Lett. 482, 169–174 (2000)
4. Breitling, R., Gilbert, D., Heiner, M., Orton, R.J.: A structured approach for the engineering of biochemical network models, illustrated for signalling pathways. Briefings in Bioinformatics 9(5), 404–421 (2008)
5. Chabrier, N., Fages, F.: Symbolic model checking of biochemical networks. In: Priami, C. (ed.) CMSB 2003. LNCS, vol. 2602, pp. 149–162. Springer, Heidelberg (2003)
6. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press, Cambridge (1999) (third printing, 2001)
7. Cho, K.-H., Shin, S.-Y., Kim, H.-W., Wolkenhauer, O., McFerran, B., Kolch, W.: Mathematical modeling of the influence of RKIP on the ERK signaling pathway. In: Priami, C. (ed.) CMSB 2003. LNCS, vol. 2602, pp. 127–141. Springer, Heidelberg (2003)
8. Donaldson, R., Gilbert, D.: A model checking approach to the parameter estimation of biochemical pathways. In: Heiner, M., Uhrmacher, A.M. (eds.) CMSB 2008. LNCS (LNBI), vol. 5307, pp. 269–287. Springer, Heidelberg (2008)
9. Fages, F., Rizk, A.: On the analysis of numerical data time series in temporal logic. In: Calder, M., Gilmore, S. (eds.) CMSB 2007. LNCS (LNBI), vol. 4695, pp. 48–63. Springer, Heidelberg (2007)
10. Gilbert, D., Heiner, M.: From petri nets to differential equations - an integrative approach for biochemical network analysis. In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 181–200. Springer, Heidelberg (2006)
11. Gilbert, D., Heiner, M., Lehrack, S.: A unifying framework for modelling and analysing biochemical pathways using petri nets. In: Calder, M., Gilmore, S. (eds.) CMSB 2007. LNCS (LNBI), vol. 4695, pp. 200–216. Springer, Heidelberg (2007)
12. Heiner, M., Gilbert, D., Donaldson, R.: Petri nets for systems and synthetic biology. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 215–264. Springer, Heidelberg (2008)
13. Hansson, H., Jonsson, B.: A Logic for Reasoning about Time and Reliability. Formal Aspects of Computing 6(5), 512–535 (1994)
14. Kolch, W., Calder, M., Gilbert, D.: When kinases meet mathematics: the systems biology of MAPK signalling. FEBS Lett. 579, 1891–1895 (2005)
15. Levchenko, A., Bruck, J., Sternberg, P.W.: Scaffold proteins may biphasically affect the levels of mitogen-activated protein kinase signaling and reduce its threshold properties. Proc. Natl. Acad. Sci. USA 97(11), 5818–5823 (2000)

16. Merz, S.: Model checking: A tutorial overview. In: Cassez, F., Jard, C., Rozoy, B., Dermot, M. (eds.) MOVEP 2000. LNCS, vol. 2067, pp. 3–38. Springer, Heidelberg (2001)
17. Matsuno, H., Fujita, S., Doi, A., Nagasaki, M., Miyano, S.: Towards biopathway modeling and simulation. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 3–22. Springer, Heidelberg (2003)
18. Murata, T.: Petri nets: Properties, analysis and applications. Proc.of the IEEE 77(4), 541–580 (1989)
19. Orton, R., Sturm, O.E., Gormand, A., Kolch, W., Gilbert, D.: Computational modelling reveals feedback redundancy within the epidermal growth factor receptor/extracellular-signal regulated kinase signalling pathway. Systems Biology 2, 173–183 (2008)
20. Pnueli, A.: The Temporal Semantics of Concurrent Programs. Theor. Comput. Sci. 13, 45–60 (1981)
21. Popova-Zeugmann, L., Heiner, M., Koch, I.: Time Petri Nets for Modelling and Analysis of Biochemical Networks. Fundamenta Informaticae 67, 149–162 (2005)
22. Schoeberl, B., Eichler-Jonsson, C., Gilles, E.D., Muller, G.: Computational modeling of the dynamics of the MAP kinase cascade activated by surface and internalized EGF receptors. Nature Biotechnology 20, 370–375 (2002)

Multilevel Modeling of Morphogenesis

Paulien Hogeweg

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

Abstract. In order to study the growth and development of cellular systems one needs a formalism in which one can combine the biophysical properties of cells with the modulation of these properties by gene regulatory processes. I will argue that the multiscale 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 a 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 a pre-eminent example of a process which bridges multiple levels of organization.

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

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

Moreover, although biological systems are highly complex the formalism 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.

2 Some Material Properties of Biological Cells

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

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 29–35, 2009.

- 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 (experimentally shown in Paramecium by Iwamoto et al., 2005).
- **Compressibility** under mechanical forces (experimentally shown by, e.g., Tricky et al. 2006: they measure a Poisson ratio of .36 in chondrocytes; volume variations up to 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 the review by 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:

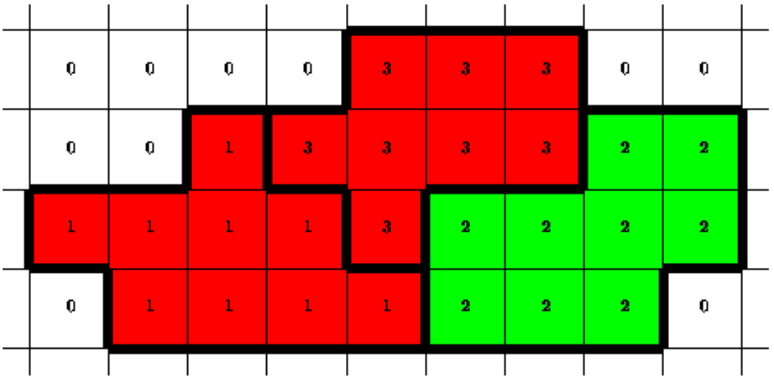


Fig. 1. Representation of biological cells in CPM

- The cells are mapped to the cellular automaton as a set of contiguous grid cells with identical state (see Fig. 1).
- 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_{ij} represent surface energy between cells (and J_{im} that between a cell and the medium).
– Copy state of neighboring cell with probability:

$$P_{i\to j} = 1 \qquad \text{iff} \quad \Delta H < -Y,$$
$$P_{i\to j} = e^{-(\Delta H+Y)/T} \qquad \text{iff} \quad \Delta H \geq -Y,$$

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

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:

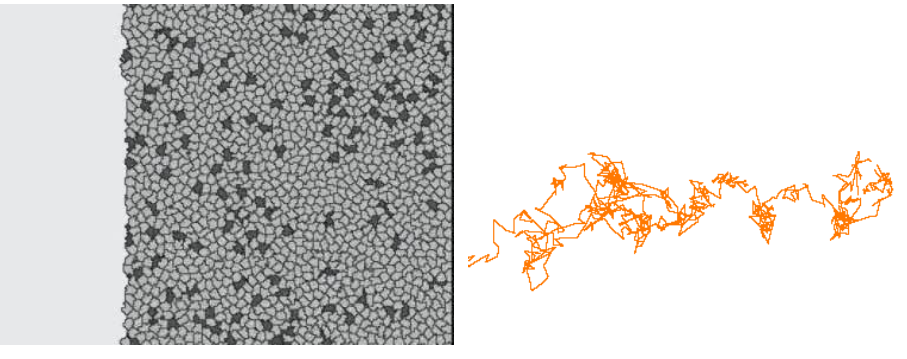


Fig. 2. Size differences may lead to movement "against the flow" (cf. Käfer et al., 2006)

– 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, and 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 information rich dynamical entities, which change their 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_{ij}) 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 by information 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 — $(J_{ij} \rightarrow J'_{ij})$
  production of morphogens
  changes in Y, V, M
- induced growth — $(V{+}{+})$
- cell division — $\sigma_i \rightarrow \sigma_i + \sigma_j$
- 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 and Hogeweg, 2001*
- chemotaxis — $(\Delta H = \Delta H + grad)$
- polarity: persistent, adjustable directional motion — *Beltman et al., 2007*
- explicit intracellular cytoskeleton dynamics — *Marée et al., 2006*
- particle based modeling of intracellular reaction dynamics — *Hogeweg and 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 of 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 another 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 for the selection, "generic" morphogenetic mechanisms could be uncovered, i.e., those which emerged as side effects 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. Then, by this model, uncovered mechanisms of morphogenesis resemble those described in multicolor

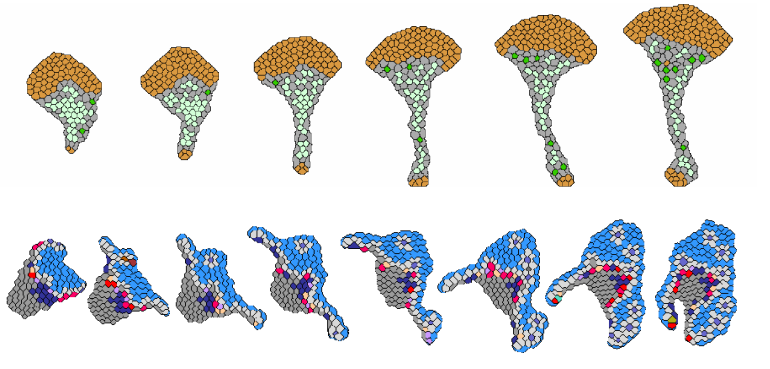


Fig. 3. Evolution for cell differentiation leads to morphogenesis (Hogeweg 2000a,b)

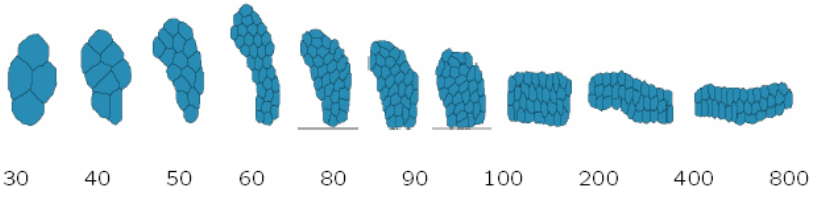


Fig. 4. Convergent extension in bipolar cells (cf Zajac et al., 2003; Hogeweg, unpublished)

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 and Gierer, 2000), or to purely information models of cells whose interactions are solely based on ancestry rather than on a dynamic neighborhood of other cells (e.g., L systems) (Lindenmayer, 1968; Hogeweg and Hesper 1974; Prusinkiewicz and Lindenmayer, 1990). The modeling formalism described here goes an important step beyond those approaches by combining generic (mechanical) properties of cells with information 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.

References

1. Beltman, J.B., Marée, A.F., Lynch, J.N., Miller, M.J., de Boer, R.J.: Lymph node topology dictates T cell migration behavior. J. Exp. Med. 204(4), 771–780 (2007)
2. Chen, C.S., Mrksich, M., Huang, S., Whitesides, G.M., Ingber, D.E.: Geometric control of cell life and death. Science 276, 1425–1428 (1997)
3. Glazier, J.A., Graner, F.: Simulation of the differential driven rearrangement of biological cells. Phys. Rev. E 47, 2128–2154 (1993)
4. Graner, F., Glazier, J.A.: Simulation of biological cell sorting using a two-dimensional extended Potts model. Phys. Rev. Lett. 69(13), 2013–2016 (1992)
5. Grieneisen, V.A., Xu, J., Marée, A.F., Hogeweg, P., Scheres, B.: Auxin transport is sufficient to generate a maximum and gradient guiding root growth. Nature 449, 1008–1013 (2007)
6. Hogeweg, P.: Evolving mechanisms of morphogenesis: on the interplay between differential adhesion and cell differentiation. J. Theor. Biol. 203(4), 317–333 (2000)
7. Hogeweg, P.: Shapes in the shadow: evolutionary dynamics of morphogenesis. Artif. Life 6(1), 85–101 (2000)
8. Hogeweg, P.: Computing an organism: on the interface between informatic and dynamic processes. BioSystems 64(1-3), 97–109 (2002)
9. Hogeweg, P., Hesper, B.: A model study on morphological description. Pattern Recogn. 6 (1974)
10. Hogeweg, P., Takeuchi, N.: Multilevel selection in models of prebiotic evolution: compartments and spatial self-organization. Orig. Life Evol. Biosph. 33(4-5), 375–403 (2003)
11. Käfer, J., Hogeweg, P., Marée, A.F.: Moving forward moving backward: directional sorting of chemotactic cells due to size and adhesion differences. PLoS Comput. Biol. 2(6) (2006)

12. Lecuit, T., Lenne, P.F.: Cell surface mechanics and the control of cell shape, tissue patterns and morphogenesis. Nat. Rev. Mol. Cell Biol. 8(8), 633–644 (2007)
13. Lindenmayer, A.: Mathematical models for cellular interactions in development. I. Filaments with one-sided inputs. J. Theor. Biol. 18(3), 280–299 (1968)
14. Marée, A.F., Hogeweg, P.: How amoeboids self-organize into a fruiting body: multicellular coordination in Dictyostelium discoideum. Proc. Natl. Acad. Sci. USA 98(7), 3879–3883 (2001)
15. Marée, A.F., Jilkine, A., Dawes, A., Grieneisen, V.A., Edelstein-Keshet, L.: Polarization and movement of keratocytes: a multiscale modelling approach. Bull. Math. Biol. 68(5), 1169–1211 (2006)
16. Marée, A.F., Panfilov, A.V., Hogeweg, P.: Migration and thermotaxis of dictyostelium discoideum slugs, a model study. J. Theor. Biol. 199(3), 297–309 (1999)
17. Meinhardt, H., Gierer, A.: Pattern formation by local self-activation and lateral inhibition. Bioessays 22(8), 753–760 (2000)
18. Mombach, J.C., Glazier, J.A., Raphael, R.C., Zajac, M.: Quantitative comparison between differential adhesion models and cell sorting in the presence and absence of fluctuations. Phys. Rev. Lett. 75(11), 2244–2247 (1995)
19. Prusinkiewicz, P., Lindenmayer, A.: The Algorithmic Beauty of Plants. Springer, New York (1990)
20. Savill, N.J., Hogeweg, P.: Modelling morphogenesis: from single cells to crawling slugs. J. Theor. Biol. 184, 229–235 (1997)
21. Segel, L.: Computing an organism. Proc. Natl. Acad. Sci. USA 98(7), 3639–3640 (2001)
22. Zajac, M., Jones, G.L., Glazier, J.A.: Simulating convergent extension by way of anisotropic differential adhesion. J. Theor. Biol. 222(2), 247–259 (2003)

A Definition of Cellular Interface Problems

Markus Kirkilionis, Mirela Domijan, Martin Eigel,
Erwin George, Mike Li, and Luca Sbano

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

Abstract. We define a class of cellular interface problems (short: CIP) that mathematically model the exchange of molecules in a compartmentalised living cell. Defining and eventually solving such compartmental problems is important for several reasons. They are needed to understand the organisation of life itself, for example by exploring different 'origin of life' hypothesis based on simple metabolic pathways and their necessary division into one or more compartments. In more complex forms investigating cellular interface problems is a way to understand cellular homeostasis of different types, for example ionic fluxes and their composition between all different cellular compartments. Understanding homeostasis and its collapse is important for many physiological medical applications. This class of models is also necessary to formulate efficiently and in detail complex signalling processes taking place in different cell types, with eukaryotic cells the most complex ones in terms of sophisticated compartmentalisation. We will compare such mathematical models of signalling pathways with rule-based models as formulated in membrane computing in a final discussion. The latter is a theory that investigates computer programmes with the help of biological concepts, like a subroutine exchanging data with the environment, in this case a programme with its global variables.

1 Setting the Problem

Most theories about the origin of life depend on the relative closedness of a reaction volume allowing for either the protected replication of a 'replicator' like primitive RNA, or the persistence of a simple metabolic pathway where without a protective and also selective membrane the reaction system would dilute or be perturbed and cease to exist (for some speculation see for example [16]). The membranes of biology are formed by lipids which have astonishing chemical properties allowing them to build so-called vesicles in a self-organised fashion and in many different environments. By either allowing the vesicles to split and preserving the metabolic pathway, or by positioning the replicator into both daughter vesicles, it is argued that this new entity is able to allow Darwinian evolution. In this case replication may not be perfect, allowing for mutations, or the metabolic pathway becomes perturbed, or both. After the subsequent unavoidable selection process the complexity in terms of 'division of labour' and

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 36–62, 2009.

therefore the possibilities of primitive life to adapt to changing environments, but also the precision of the replication process itself might have gradually increased.

Such speculations about different possible paths to produce stable life, i.e cells that are able to reproduce, that have a metabolism (use energy to decrease their own entropy) and are able to adapt, moreover the ability to evolve on larger time scales, enable us to better understand the organisatorial prerequisites for the working of even the simplest present cellular signalling pathway. Compartmental organisation in other words is defining the necessary system boundaries or form without which the signalling system and other cellular function would never have come into existence.

It is fruitful to approach problems in computer science with similar biological ideas and concepts. Such cross-over is called natural computing, with membrane computing being an important sub-discipline. We will introduce another type of computational device, a density approach to molecular concentrations inside the cell. In other words we will make use of continuous distributions of molecules, so go from a discrete to a continuous perspective. It is tempting to compare the latter with ideas from membrane computing.

2 A Modular Approach

The problem which we will call 'Cellular Interface Problem' (CIP) is constructed with the help of partial differential equations (PDE) defined on the cell volumes, and interface or boundary conditions defined on the membranes, depending whether a membrane is a system boundary, or whether the membrane is enclosing a compartment internal to the system. The state of the system is characterised by concentrations of molecules, again either defined on compartmental volumes or as concentrations on the interfaces. Clearly what we have in mind is a direct correspondence between a density of molecules that can be directly measured in a cell, for example with the help of a confocal microscope. We will come back to this point in an own section. In this view the system state is very closely chosen to represent measurable quantities, i.e. can be called 'empirical'. We note that 'empirical' necessarily can only be defined according to a given spatial scale, which here is the one of the microscope. We are aiming at simulating the time course of concentration changes in the cell which if predicted correctly, i.e. if there is a correspondence between measured data and simulated data, can tell us something about the working of the 'real' cell. The system state is a function of time t and space x, so is spatially explicit. We are making this choice because we are interested in transport processes not only across the membrane, but also inside each cellular compartment. Many more examples and simulations related to this definition can be found in [14].

2.1 Events

Any model describing changes in molecular distribution and reaction between species is necessarily event driven. Here the basic events are either molecular displacements, numbers of reactions between species of molecules, or conformational changes of molecules (like proteins), all relative to a given time scale

(interval) $\delta > 0$ assumed to be small but sufficiently large such that at least one displacement or reaction/conformational change is expected to happen. There is a universal clock assumed for all these events, and all possible events can take place in parallel. At the given typical physical scales related to the microscope this event structure is assumed to be close to reality.

2.2 Membrane System

The membrane system of a biological cell consists of a lipid bilayer. Its chemical properties determine which molecules can pass this membrane spontaneously, or need help in passing it. This might require energy and is then called active transport. There are other forms of transport in biological cells, for example vesicles budding from the membrane and fusing to a membrane at a different location (secretory pathway). But this will not be considered in the following as this would mean we could not work with a fixed geometry of the membrane system. Here we consider a fixed general setting, for simplicity in two spatial dimensions only. The outer membrane (cell wall) is represented by a piecewise smooth boundary denoted by Γ_c, see Fig.1. The volume encircled by Γ_c and representing the cytoplasm is given by Ω_c. There are possibly nested compartments lying entirely inside Ω_c. The largest is typically the cell nucleus, and the mathematical counterpart of the nuclear envelope is denoted by Γ_n, the volume encircled by Ω_n. There are possibly different substructures inside Ω_n, not necessarily modelling a lipid bilayer, but definitely an area with different properties of the medium where a molecule might have to pass in a different way. Such areas are modelled again by subdomains Ω_i, $i = 1, \ldots, s$, with corresponding boundaries Γ_i, $i = 1, \ldots, s$.

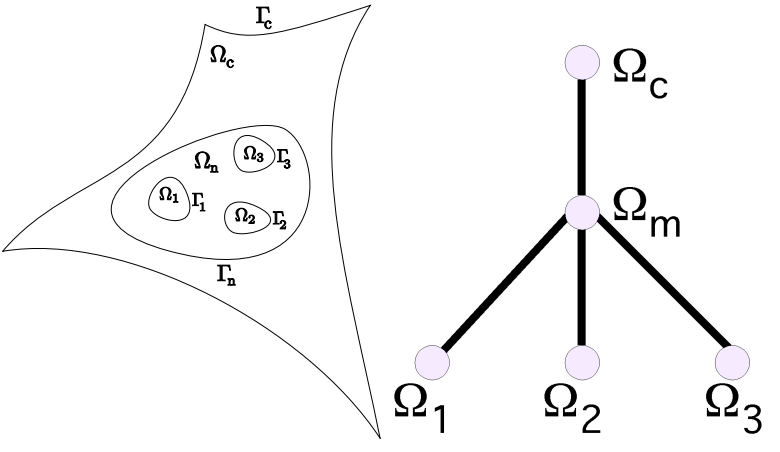


Fig. 1. A compact smooth domain $\overline{\Omega_c}$ with interior Ω_c and sub-subdomains $\overline{\Omega}_i$, $i = 1, \ldots, s$ inside a subdomain $\overline{\Omega_n}$ of Ω_c. The right graph shows the hierarchy of the domain data-structure of the example.

2.3 Ion Channels and Transporters

We are interested to consider membranes which are permeable and which allow molecules to pass from one compartment to another. Biologically one way of transport is through channels which are here considered as *molecular machines* constituted by macromolecules which can be in different conformations. Such macromolecules allow to transfer smaller molecules from one membrane side to the other, often through a regulated mechanism. Experiments show that the movement of a molecule through the channel can be considered sequentially and discretely, often corresponding to conformational changes. The nature of the steps is dictated by the molecular interactions, this implies that from the microscopic point of view the transitions form a stochastic process. In the following we consider some fundamental structure of such a process. We spend some time on this example because it will explain very easily why we need to introduce temporal and spatial scales into the derivation of a CIP. A channel can be described as a cell compartment with a certain number of internal sub-compartments. Let this number be m. If we want to include the two exits we can say that the total number is $m + 2$. A picture of this can be see in Fig.2. A molecule crossing the channel has to move from compartment 0 to compartment $m+1$. The transitions are in general reversible.

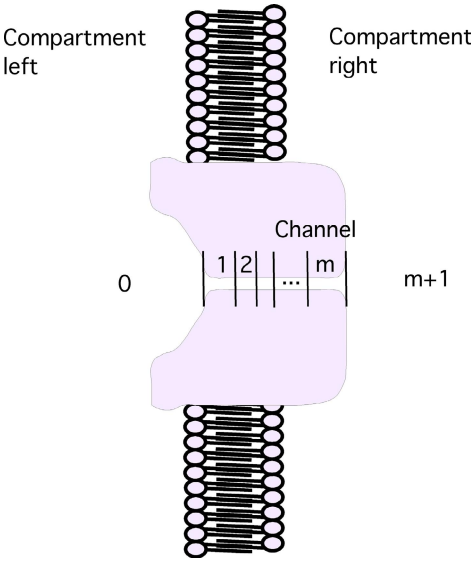


Fig. 2. A schematic view of a channel in a membrane. Compartments $left$ and $right$ are external to the channel.

Now let us observe that the state of the channel can be described by considering the occupation of one of the $m + 2$ sub-compartments. We now make the crucial simplifying assumption that only one single molecule can cross the channel at one time. This allows us to define the state space of the channel as

$$S \doteq \{i \in \{0, ..., m+2\}\}. \tag{1}$$

Now any motion in the channel of a molecule in the direction from A to B (see in Fig.2) will correspond to a sequence of transitions in S of the form

$$i \to i+1.$$

Similarly any motion of a molecule from B to A (see in Fig.2) will correspond to a sequence of transitions of the type

$$i \to i-1.$$

Since the transitions are stochastic such a structure is naturally described through the notion of a Markov Chain (MC). On S the Markov chain is constructed by determining the transition probabilities

$$P(s = i|s = j) = p_{ij} \text{ with } i, j \in S.$$

The transition probabilities satisfy the Markov property:

$$p_{i_1 i_3} = \sum_{i_2 \in S} p_{i_1 i_2} p_{i_2 i_3}. \tag{2}$$

One can easily note that since transition are possible only through adjacent sub-compartments we have that

$$p_{ij} = 0 \text{ for } j > i+1 \text{ and } j < i-1.$$

The natural way to construct the transition probability matrix $P = (p_{ij})$ is to consider its infinitesimal generator defined as

$$K = \lim_{t \to 0} \frac{P(t) - \mathbf{I}}{t}. \tag{3}$$

Let $p_i(t)$ be the probability that a time t the channel is in state i, then the time evolution of the Markov chain is governed by

$$\frac{dp_i(t)}{dt} = \sum_{j \in S} p_i(t)\, K_{ji}. \tag{4}$$

Looking at the scheme in Fig.2 we can encode the Markov chain in a graph, the so-called *Interaction Graph*.

Each vertex in the graph represents a sub-compartment and the arrows are drawn according to the following rule:

$$\text{There is an arrow from } i \text{ to } j \text{ if and only if } K_{ij} \neq 0. \tag{5}$$

Recall that in any Markov chain $K_{ii} = -\sum_{j \neq i} K_{ij}$ and $K_{ij} \geq 0$ for any $i \neq j$. We should notice that in the example in Fig.3 the channel is one way: a molecule that enters the channel in 0 eventually will reach the other end $m+1$.

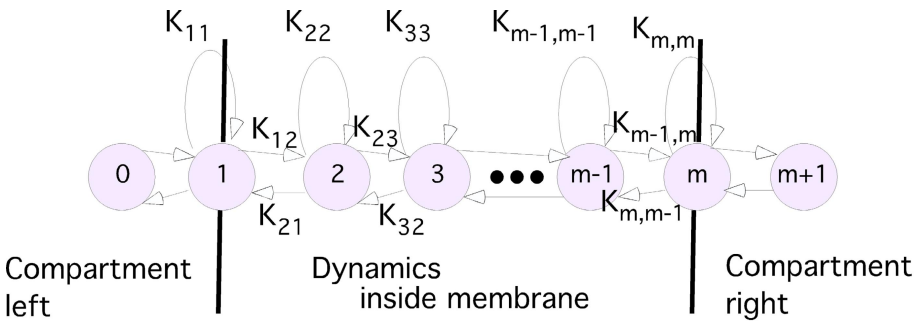


Fig. 3. The channel interaction graph

From the modelling point of view the matrix K is formed by the rates at which in unit time transitions occur. Such rates can be derived by statistical mechanics arguments. Finally recall a crucial property of Markov chains: ergodicity. For our purpose we are interested in observing that for large times ($t \to \infty$) equation (4) a steady state solution given by solving:

$$p\,K = 0 \text{ or } K^T p = 0. \tag{6}$$

The solution of (6) can be normalised and it is called *invariant measure*. Other transporters establishing for example a so-called *symport* or *antiport* can be modelled in a similar way. The concept of assigning a graph to molecular transitions, with the interaction graph being the most basic one, can be found in [26], and for mass-action reaction systems in [10].

2.4 Fluxes across Membranes and Movement Inside Compartments

In the following we restrict our attention to derive flux conditions across just a single membrane. Let therefore Ω now just be a two dimensional domain divided into two sub-domains Ω_1, Ω_2 such that

$$\Omega = \Omega_1 \cup \Omega_2.$$

The membrane is geometrically represented by the common boundary of the two sub-domains

$$\Gamma = \Omega_1 \cap \Omega_2.$$

Fluxes across Membranes Derived from Microscopic Channel Dynamics. In the membrane (interface) Γ we assume there are channels which can be open for molecules to cross the membrane, as introduced in the previous section. For simplicity let us assume there is a simple diffusion process inside each compartment described by the standard diffusion equation in both Ω_1 and Ω_2. Each of the channels in Γ is described independently by an m-state Markov chain whose generator is $K(y)$, where $y \in \Gamma$. Then the diffusion scaling $\delta^2/\tau = D$ allows two possible boundary conditions at Γ. Let ρ_i be the molecular density in Ω_i and the membrane be given according to the convention

$$\Gamma = \{(x, y) \in \Omega_1 \cup \Omega_2 : x = 0\}.$$

As we will motivate in the following there will be two different interface conditions that are induced by the channel dynamics, and which are related to different relative time-scales of channel transitions with respect to transport inside a compartment of a given molecule. By using the interface Γ coordinates, and by denoting with $\pi(y) = (\pi_1(y), \ldots, \pi_m(y))$ the transition probabilities of all channels located at $y \in \Gamma$ (we used the notation p before for a single channel), these different interface conditions are:

1. If the Markov chain evolves on a time scale equal to the diffusion then the boundary condition is

$$\rho_1(t, 0, y) = \pi_1(t, y), \quad \rho_2(t, 0, y) = \pi_m(t, y),$$

and

$$\begin{aligned} \frac{\partial \rho_1(t,0,y)}{\partial x} &= \frac{1}{D}\left(k_{12}(y)\rho_1(t,0,y) - k_{21}(y)\pi_2(t,y)\right), \\ \frac{\partial \rho_2(t,0,y)}{\partial x} &= \frac{1}{D}\left(-k_{m-1,m}(y)\rho_2(t,0,y) + k_{m,m-1}(y)\pi_{m-1}(t,y)\right). \end{aligned} \tag{7}$$

As we consider our previous channel example embedded into a spatial context it should be noted that the entries of K are nonzero only in the diagonal and the two off-diagonals. Furthermore it was assumed that at the 'left' gate we have $K_{11} = -\frac{k_{12}}{\delta}$, $K_{12} = \frac{k_{12}}{\delta}$, $K_{21} = \frac{k_{21}}{\delta}$ and $K_{22} = -\frac{k_{21}}{\delta} - k_{23}$, with the k_{ij} being given positive transition rates characterising the channel (see Fig.3), and δ a spatial scale. Symmetric assumptions have been made for transition rates at the 'right' channel gate. The spatial scale δ will be the grid size of a grid introduced to perform the continuum limit of a transport process inside the volume of the compartments. This setting of the transition rates might be of course different for different kinds of transport through the membrane that do not follow the discrete diffusion logic. Here $\pi_1(t, y)$, and $\pi_m(t, y)$ are the probability distributions on the two exits of the channel at $(0, y)$. Their dynamics is given by the ODE

$$\frac{d\pi_\alpha(t, y)}{dt} = \sum_{\beta=1}^{m} K_{\alpha\beta}(y)\pi_\beta(t, y), \quad \alpha = 2, ..., m-1.$$

These conditions as the ones below should hold for all $y \in \Gamma$, and $t \in [t_0, T]$, with t_0 the start of the experiment, and T the time where the experiments or at least its observation stops.

2. If the Markov chain evolves on a time scale faster than the diffusion then the boundary condition is

$$\rho_1(t, 0, y) = \mu_1(y), \quad \rho_2(t, 0, y) = \mu_m(y),$$

where $\mu(y) = (\mu_1(y), \ldots, \mu_m(y))$ is the invariant measure of the Markov chain modelling the channel.

We now motivate the derivation of the interface conditions given above. Let us partition Ω into a a two dimensional lattice $\Lambda_\delta = (\delta\mathbb{Z})^2$, with $\delta > 0$ being the fundamental length of the lattice Λ_δ. Clearly as $\delta \to 0$ the lattice tends to $\mathbb{R}^2$. Each point in Λ_δ is identified by a couple of coordinates (n_x, n_y). We denote by $P_i(t, n_x, n_y)$ the probability that a particle is in the site (n_x, n_y) at time t. Let $\tau > 0$ be a given time scale. We consider the following discrete diffusion process in Ω_i:

$$P_i(t+\tau, n_x, n_y) = \frac{1}{2}\mathbf{D}_x(P_i(t, n_x, n_y)) + \frac{1}{2}\mathbf{D}_y(P_i(t, n_x, n_y),) \tag{8}$$

where $\mathbf{D}_x$ and $\mathbf{D}_y$ are operators defined by

$$\mathbf{D}_x f(n_x, n_y) \doteq f(n_x + 1, n_y) + f(n_x - 1, n_y), \tag{9}$$

$$\mathbf{D}_y f(n_x, n_y) \doteq f(n_x, n_y + 1) + f(n_x, n_y - 1). \tag{10}$$

Without loss of generality the membrane Γ can be parameterised by

$$\Gamma = \{(n_x, n_y) \in \Lambda_\delta : n_x = 0\}.$$

On the membrane the diffusion process implies that

$$P_i(t+\tau, 0, n_y) = P_i(t, -1, n_y) + \frac{1}{2}\mathbf{D}_y(P_i(t, 0, n_y)). \tag{11}$$

Now we introduce the channels into Γ. At each point $(0, n_y) \in \Gamma$ we assume that there exists a channel who has a certain number of internal m states as before (see Figure 2). Let C be the (discrete) state space of a channel. At the moment we do not enter into the details of the mechanism but simply assume that each channel is described by a Markov chain of the form

$$\Pi_\alpha(t+\tau, 0, n_y) = \Pi_\alpha(t, 0, n_y) + \tau \sum_{\beta=1}^{m} K_{\alpha\beta}(\delta, n_y)\Pi_\beta(t, n_y). \tag{12}$$

The function $\Pi_\alpha(t, 0, n_y)$ is the probability that a single molecule is in $(0, n_y) \in \Lambda_\delta$ at time t, and in the state α of the channel. By continuity we need to require the following boundary conditions:

$$P_1(t, 0, n_y) = \Pi_1(t, 0, n_y), \quad P_2(t, 0, n_y) = \Pi_m(t, 0, n_y). \tag{13}$$

Using (13) and substituting (12) into (11) we obtain:

$$P_i(t, 0, n_y) + \tau \sum_{\beta=1}^{m} K_{i\beta}(\delta, n_y)\Pi_\beta(t, n_y) = P_i(t, -1, n_y)) + \frac{1}{2}\mathbf{D}_y(P_i(t, 0, n_y)). \tag{14}$$

Next we take the continuum limit to obtain densities, denoted by ρ:

$$P_i(t, x/\delta, y/\delta) = \rho_{i,\delta}(t, x, y), \tag{15}$$

$$\Pi_1(t, y/\delta) = \rho_{1,\delta}(t, 0, y), \tag{16}$$

$$\Pi_m(t, y/\delta) = \rho_{2,\delta}(t, 0, y), \tag{17}$$

$$\Pi_\alpha(t, y/\delta) = \pi_{\alpha,\delta}(t, y) \text{ for } \alpha \neq 1, m. \tag{18}$$

Note that $\pi_{\alpha,\delta}(t,y)$ can be interpreted as the number of channels in state α present in a membrane segment of length δ in Γ. Making the substitution in (8) and taking the limit $\delta, \tau \to 0$ with $\frac{\delta^2}{\tau} = D > 0$ we obtain by a standard calculation that

$$\frac{\partial \rho_i(t,x,y)}{\partial t} = D\nabla^2 \rho_i(t,x,y), \quad i = 1,2. \tag{19}$$

For the boundary condition, retaining δ, τ greater than zero, we obtain

$$\rho_{i,\delta}(t+\tau,0,y) - \rho_{1,\delta}(t,0,y) = \tau\, K_{11}(\delta,y)\rho_{1,\delta}(t,0,y) + \tau\, K_{1m}(\delta,y)\rho_{2,\delta}(t,0,y) + \\ + \tau \sum_{\beta \neq 1,m} K_{1\beta}(\delta,y)\pi_{\beta,\delta}(t,y),$$

$$\rho_{m,\delta}(t+\tau,0,y) - \rho_{m,\delta}(t,0,y) = \tau\, K_{m1}(\delta,y)\rho_{1,\delta}(t,0,y) + \\ + \tau\, K_{mm}(\delta,y)\rho_{2,\delta}(t,0,y) + \tau \sum_{\beta \neq 1,m} K_{1\beta}(\delta,y)\pi_{\beta,\delta}(t,y),$$

$$\pi_{\alpha}(t+\tau,y) - \pi_{\alpha}(t,y) = \tau\, K_{\alpha 1}(\delta,y)\rho_{1,\delta}(t,0,y) + \tau\, K_{\alpha m}(\delta,y)\rho_{2,\delta}(t,0,y) + \\ + \tau \sum_{\beta=1}^{m} K_{\alpha\beta}(\delta,y)\pi_{\beta,\delta}(t,y) \text{ for } \alpha \neq 1,m,$$

and

$$\rho_{1,\delta}(t,0,y) - \rho_{1,\delta}(t,-\delta,y) = -\tau\, K_{11}(\delta,y)\rho_{1,\delta}(t,0,y) - \tau\, K_{1m}(\delta,y)\rho_{2,\delta}(t,0,y) + \\ - \tau \sum_{\beta \neq 1,m} K_{1\beta}(\delta,y)\pi_{\beta,\delta}(t,y) + \frac{1}{2}(\rho_{1,\delta}(t,0,y+\delta) + \rho_{1,\delta}(t,0,y-\delta)),$$

$$\rho_{2,\delta}(t,0,y) - \rho_{2,\delta}(t,-\delta,y) = -\tau\, K_{m1}(\delta,y)\rho_{1,\delta}(t,0,y) - \tau\, K_{mm}(\delta,y)\rho_{2,\delta}(t,0,y) + \\ - \tau \sum_{\beta \neq 1,m} K_{m\beta}(\delta,y)\pi_{\beta,\delta}(t,y) + \frac{1}{2}(\rho_{2,\delta}(t,0,y+\delta) + \rho_{2,\delta}(t,0,y-\delta)).$$

These expression can be further simplified by considering that $K_{1m}(\delta,y) = K_{m1}(\delta,y) = 0$, because the two ends on the channel do not communicate directly. The new conditions read

$$\rho_{1,\delta}(t+\tau,0,y) - \rho_{1,\delta}(t,0,y) = \tau\, K_{11}(\delta,y)\rho_{1,\delta}(t,0,y) + \tau \sum_{\beta \neq 1,m} K_{1\beta}(\delta,y)\pi_{\beta,\delta}(t,y),$$

$$\rho_{m,\delta}(t+\tau,0,y) - \rho_{m,\delta}(t,0,y) = \tau\, K_{mm}(\delta,y)\rho_{2,\delta}(t,0,y) + \\ + \tau \sum_{\beta \neq 1,m} K_{1\beta}(\delta,y)\pi_{\beta,\delta}(t,y),$$

$$\pi_{\alpha,\delta}(t+\tau,y)-\pi_{\alpha,\delta}(t,y)=\tau\,K_{\alpha 1}(\delta,y)\rho_{1,\delta}(t,0,y)+\tau\,K_{\alpha m}(\delta,y)\rho_{2,\delta}(t,0,y)+$$
$$+\tau\sum_{\beta=1}^{m}K_{\alpha\beta}(\delta,y)\pi_{\beta,\delta}(t,y)\text{ for }\alpha\neq 1,m,$$

and

$$\rho_{1,\delta}(t,0,y)-\rho_{1,\delta}(t,-\delta,y)=-\tau\,K_{11}(\delta,y)\rho_{1,\delta}(t,0,y)+$$
$$-\tau\sum_{\beta\neq 1,m}K_{1\beta}(\delta,y)\pi_{\beta,\delta}(t,y)+\frac{1}{2}(\rho_{1,\delta}(t,0,y+\delta)+\rho_{1,\delta}(t,0,y-\delta)),$$

$$\rho_{2,\delta}(t,0,y)-\rho_{2,\delta}(t,-\delta,y)=-\tau\,K_{mm}(\delta,y)\rho_{2,\delta}(t,0,y)+$$
$$-\tau\sum_{\beta\neq 1,m}K_{m\beta}(\delta,y)\pi_{\beta,\delta}(t,y)+\frac{1}{2}(\rho_{2,\delta}(t,0,y+\delta)+\rho_{2,\delta}(t,0,y-\delta)).$$

To simplify these conditions let us consider

$$\pi_{\alpha,\delta}(t+\tau,y)-\pi_{\alpha,\delta}(t,y)=\tau\frac{\partial\pi_{\alpha,\delta}(t,y)}{\partial t}+o(\tau),$$

$$\rho_{i,\delta}(t+\tau,0,y)-\rho_{i,\delta}(t,0,y)=\tau\frac{\partial\rho_{i,\delta}(t,0,y)}{\partial t}+o(\tau),$$

$$\rho_{i,\delta}(t,0,y)-\rho_{i,\delta}(t,-\delta,y)=\delta\frac{\partial\rho_{i,\delta}(t,0,y)}{\partial x}+o(\delta),$$

and

$$\rho_{i,\delta}(t,0,y+\delta)+\rho_{i,\delta}(t,0,y-\delta)=o(\delta).$$

Using the previous approximation the boundary condition can finally be rewritten as

$$\begin{aligned}
&\frac{\partial\rho_{1,\delta}(t,0,y)}{\partial t}=K_{11}(\delta,y)\rho_{1,\delta}(t,0,y)+\sum_{\beta\neq 1,m}K_{1\beta}(\delta,y)\pi_{\beta,\delta}(t,y),\\
&\frac{\partial\rho_{2,\delta}(t,0,y)}{\partial t}=K_{mm}(\delta,y)\rho_{2,\delta}(t,0,y)+\sum_{\beta\neq 1,m}K_{1\beta}(\delta,y)\pi_{\beta,\delta}(t,y),\\
&\frac{\partial\pi_{\alpha,\delta}(t,y)}{\partial t}=K_{\alpha 1}(\delta,y)\rho_{1,\delta}(t,0,y)+K_{\alpha m}(\delta,y)\rho_{2,\delta}(t,0,y)\\
&+\textstyle\sum_{\beta=1}^{m}K_{\alpha\beta}(\delta,y)\pi_{\beta,\delta}(t,y)\text{ for }\alpha\neq 1,m,\\
&\frac{\partial\rho_{1,\delta}(t,0,y)}{\partial x}=-\frac{\tau}{\delta}\,K_{11}(\delta,y)\rho_{1,\delta}(t,0,y)-\frac{\tau}{\delta}\sum_{\beta\neq 1,m}K_{1\beta}(\delta,y)\pi_{\beta,\delta}(t,y),\\
&\frac{\partial\rho_{2,\delta}(t,0,y)}{\partial x}=-\frac{\tau}{\delta}\,K_{mm}(\delta,y)\rho_{2,\delta}(t,0,y)-\frac{\tau}{\delta}\sum_{\beta\neq 1,m}K_{m\beta}(\delta,y)\pi_{\beta,\delta}(t,y).
\end{aligned}\tag{20}$$

In the following we want to take the limit $\delta, \tau \to 0$ while preserving the diffusive scale $\frac{\delta^2}{\tau} = D$. By inspection of equations (20) one can note that the limits that need to be studied are

$$\lim_{\delta \to 0} K_{\alpha\beta}(\delta, y), \tag{21}$$

and

$$\lim_{\delta \to 0} \frac{\tau}{\delta} K_{\alpha\beta}(\delta, y) = \lim_{\delta \to 0} \frac{\delta}{D} K_{\alpha\beta}(\delta, y). \tag{22}$$

The limits (21) and (22) cannot be both finite and different from zero at the same time. Therefore we have two cases, as introduced in the beginning of this subsection:

(Case A): Suppose that

$$\lim_{\delta \to 0} K_{\alpha\beta}(\delta, y) = K_{\alpha\beta}(y). \tag{23}$$

Clearly we have

$$\lim_{\delta \to 0} \pi_\delta(t, y) = \pi(t, y) = (\pi_1(t, y), \ldots, \pi_m(t, y))$$

and in the two compartments the two densities

$$\rho_1(t, x, y), \quad \rho_2(t, x, y)$$

are defined. They satisfy

$$\rho_1(t, 0, y) = \pi_1(t, y), \quad \rho_2(t, x, y) = \pi_4(t, y)$$

by continuity at the boundary. We just consider next the Neumann part of the interface conditions. When we apply the general equations (20) to our specific channel setting, moreover using the diffusion scaling $D = \frac{\delta^2}{\tau}$, we get

$$\begin{aligned} \frac{\partial \rho_1(t, 0, y)}{\partial x} &= \frac{1}{D} \left(k_{12}(y) \rho_1(t, 0, y) - k_{21}(y) \pi_2(t, y) \right), \\ \frac{\partial \rho_2(t, 0, y)}{\partial x} &= \frac{1}{D} \left(-k_{m-1,m}(y) \rho_2(t, 0, y) + k_{m,m-1}(y) \pi_3(t, y) \right). \end{aligned} \tag{24}$$

(Case B): Suppose that

$$\lim_{\delta \to 0} \frac{\delta}{D} K_{\alpha\beta}(\delta, y) = \frac{1}{D} \tilde{K}_{\alpha\beta}(y). \tag{25}$$

This implies that

$$K_{\alpha\beta}(\delta, y) \simeq \frac{1}{\delta} \tilde{K}_{\alpha\beta}(y).$$

So the boundary conditions are of the form

$$\begin{aligned}
\frac{\partial \rho_{1,\delta}(t,0,y)}{\partial t} &= \frac{1}{\delta}\tilde{K}_{11}(y)\rho_{1,\delta}(t,0,y) + \frac{1}{\delta}\sum_{\beta\neq 1,m}\tilde{K}_{1\beta}(y)\pi_{\beta,\delta}(t,y),\\
\frac{\partial \rho_{2,\delta}(t,0,y)}{\partial t} &= \frac{1}{\delta}\tilde{K}_{mm}(y)\rho_{2,\delta}(t,0,y) + \frac{1}{\delta}\sum_{\beta\neq 1,m}\tilde{K}_{1\beta}(y)\pi_{\beta,\delta}(t,y),\\
\frac{\partial \pi_{\alpha,\delta}(t,y)}{\partial t} &= \frac{1}{\delta}\tilde{K}_{\alpha 1}(y)\rho_{1,\delta}(t,0,y) + \frac{1}{\delta}\tilde{K}_{\alpha m}(y)\rho_{2,\delta}(t,0,y) +\\
&\quad +\tfrac{1}{\delta}\textstyle\sum_{\beta=1}^{m}\tilde{K}_{\alpha\beta}(y)\pi_{\beta,\delta}(t,y) \text{ for } \alpha\neq 1,m,\\
\frac{\partial \rho_{1,\delta}(t,0,y)}{\partial x} &= -\frac{1}{D}\tilde{K}_{11}(y)\rho_{1,\delta}(t,0,y) - \frac{1}{D}\sum_{\beta\neq 1,m}\tilde{K}_{1\beta}(y)\pi_{\beta,\delta}(t,y),\\
\frac{\partial \rho_{2,\delta}(t,0,y)}{\partial x} &= -\frac{1}{D}\tilde{K}_{mm}(y)\rho_{2,\delta}(t,0,y) - \frac{1}{D}\sum_{\beta\neq 1,m}\tilde{K}_{m\beta}(y)\pi_{\beta,\delta}(t,y).
\end{aligned} \tag{26}$$

For $\delta \to 0$ the first three equations can be solved by singular perturbation theory. The leading order solution is $\tilde{K}(y)\mu(y) = 0$ where $\mu(y)$ is the invariant measure of the Markov chain describing the channel. Therefore the interface conditions we get are

$$\begin{aligned}
&\rho_1(t,0,y) = \mu_1(y),\\
&\pi_\alpha(t,0,y) = \mu_\alpha(y) \text{ for } \alpha = 2...m-1,\\
&\rho_2(t,0,y) = \mu_m(y).
\end{aligned} \tag{27}$$

Note that the choice (27) solves also the last two equations in (26).

Other Classical Boundary Conditions across Membranes. In case same relatively small molecules can just cross a lipid bilayer this can be modelled in a PDE setting by so-called Robin boundary conditions without the need to introduce an up-scaling step as we have done before in case of a simple channel dynamics. Consider again some piece of membrane denoted by Γ. Then this condition is given by

$$a(t,y)\frac{\partial}{\partial \nu}\rho + b(t,y)\rho = c(t,y) \text{ on } [0,T]\times\Gamma, \tag{28}$$

where ν is the outward pointing normal vector (we have now assumed that the piece of membrane can be curved, whereas before Γ without loss of generality was considered to be a piece of straight vertical line), and a, b and c are continuous and sufficiently smooth functions characteristic for the membrane and the molecules we are considering. The concentration ρ is considered to describe the molecular distributions inside a compartment, whereas the outer compartment concentration is assumed to be constant and therefore not considered as a state variable.

Transport Inside Compartments. Let Ω now denote the volume (here area, as we are only considering the two-dimensional case) of some cellular compartment. The molecules will move inside the compartment according to some

stochastic process. Without doing the up-scaling here explicitly, but indeed following very much the considerations about the channel dynamics in our detailed previous section on channel dynamics, the resulting equation on a macroscopic scale (here assumed to be the scale of the microscope) can lead to the following equations:

$$\frac{\partial}{\partial t}\rho(t,x) - D\triangle\rho(t,x) = 0 \ \text{ on } \ [0,T]\times\Omega. \tag{29}$$

This is the simplest case, i.e. here we would consider isotropic standard diffusion, with a diffusion coefficient $D > 0$. The operator denoted by $\triangle = \nabla^2$ is the so-called Laplace operator already used before in equation (19). Such an equation would be the limit equation resulting from Brownian motion. There are more sophisticated processes possible. One assumption could be that the molecules get stuck from time to time anywhere in Ω which can be modelled by introducing a mobile and an immobile sub-population of molecules:

$$\begin{aligned}\partial_t\rho_{mobile}(t,x) &= D\Delta\rho_{mobile}(t,x)\\ &\quad + k_b\ \rho_{immobile}(t,x) - k_d\ \rho_{mobile}(t,x) \quad \text{on } [0,T]\times\Omega \qquad (30)\\ \partial_t\rho_{immobile}(t,x) &= -k_b\ \rho_{immobile}(t,x) + k_d\ \rho_{mobile}(t,x) \quad \text{on } [0,T]\times\Omega \qquad (31)\end{aligned}$$

The positive constants k_b and k_d determine the rated at which 'bound' molecules are released and enter the mobile fraction, or get caught and are not able to diffuse any more. Clearly the observed molecular concentration as observed by the microscope would be given as the sum of the mobil and immobile fraction, i.e. $\rho = \rho_{mobile} + \rho_{immobile}$. There are of course other mathematical ways to model such 'sticky' behaviour, Again one can first introduce certain stochastic processes (different from Brownian motion) and derive an effective equation by up-scaling. This can lead to equations with so called fractional diffusion operators, giving rise to sub-diffusive behaviour. There can be directed movement of molecules as well, for example given by the fact that transporter molecules can actively transport other molecules along the cytoskeletton of the cell.

3 Combining the Modules

In order to obtain a complete meaningful specific molecular distribution model, i.e. one that eventually can be compared with data, we need to ascribe boundary and interface conditions to each of the boundaries/interfaces Γ_c, Γ_n and Γ_i, $i = 1,2,3$, see Fig. 1, and transport operators for each of the domains Ω_c, Ω_n and Ω_i, $i = 1,2,3$. Of course the same holds in case of a more general membrane system given by a tree describing the hierarchy of any system of nested subdomains. In the following we give just one simple possible example of such a complete model, but emphasise the modular character we have introduced. In reality biological knowledge and insight will lead to different hypothesis about movement and translocation of specific molecules across membranes, leading to different transport operators defined inside the compartments, and boundary and interface conditions between the compartments. Our example model is:

We assume that at Γ_c we can impose from outside the cell a fixed molecular concentration for some time:

$$\rho_c(t,y) = c(t,y) \text{ on } [0,t^*] \times \Gamma_c, \tag{32}$$

$$\frac{\partial}{\partial \nu}\rho_c(t,y) = 0 \text{ on } [t^*,T] \times \Gamma_c, \tag{33}$$

$$\partial_t \rho_c(t,x) = D\Delta\rho_c(t,x) \text{ in } [0,T] \times \Omega_c, \tag{34}$$

At Γ_n we assume channel dynamics:

$$\rho_c(t,y) = \pi_1(t,y), \quad \rho_{n,mobile}(t,y) = \pi_m(t,y), \text{on } [0,T] \times \Gamma_n, \tag{35}$$

$$\frac{\partial \rho_c(t,y)}{\partial \nu} = \frac{1}{D}\left(k_{12}(y)\rho_c(t,y) - k_{21}(y)\pi_2(t,y)\right), \text{ on } [0,T] \times \Gamma_n, \tag{36}$$

$$\frac{\partial \rho_{n,mobile}(t,y)}{\partial \nu} = \frac{1}{D}\left(-k_{m-1,m}(y)\rho_n(t,y) + k_{m,m-1}(y)\pi_{m-1}(t,y)\right), \tag{37}$$

on $[0,T] \times \Gamma_n$,

$$\frac{d\pi_i(t,y)}{dt} =$$

$$k_{i-1,i}(y)\pi_{i-1}(t,y) - (k_{i,i-1}(y) + k_{i,i+1}(y))\pi_i(t,y) + k_{i+1,i}(y)\pi_{i+1}(t,y), \tag{38}$$

$i = 2, ..., m-1$, on $[0,T] \times \Gamma_n$,

Inside Ω_n we assume molecules can possibly bind to some structure:

$$\partial_t \rho_{n,mobile}(t,x) = D\Delta\rho_{n,mobile}(t,x)$$
$$+k_b(x)\ \rho_{n,immobile}(t,x) - k_d(x)\ \rho_{n,mobile}(t,x) \text{ in } [0,T] \times \Omega_n, \tag{39}$$

$$\partial_t \rho_{n,immobile}(t,x) = -k_b(x)\ \rho_{n,immobile}(t,x) + k_d(x)\ \rho_{n,mobile}(t,x)$$
$$\text{on } [0,T] \times \Omega_n, \tag{40}$$

Inside Ω_n we assume there are some subdomains where diffusing molecules will not be able to penetrate:

$$\frac{\partial}{\partial \nu}\rho_{n,mobile}(t,y) = 0 \text{ on } [0,T] \times \Gamma_i, i = 1,2,3. \tag{41}$$

The model still needs to be complemented with initial conditions. The state of the system is given by the molecular concentrations $\rho_c(t,x)$, $\rho_n(t,x)$, and the states $\pi(t,y) = (\pi_1(t,y), \ldots, \pi_m(t,y))$, where by continuity $\pi_1(t,y) = \rho_c(t,y)$ and $\pi_m(t,y) = \rho_{n,mobile}(t,y)$ for $y \in \Gamma_n$ and al $t > 0$. The initial conditions can then be written as:

(i) Initial conditions for concentrations in compartments:

$$\rho_c(t,x) = \rho_{c,0}(x) \text{ on } \overline{\Omega}_c, \tag{42}$$

$$\rho_{n,mobile}(t,x) = \rho_{n,mobile,0}(x) \text{ on } \overline{\Omega}_n, \tag{43}$$

$$\rho_{n,immobile}(t,x) = \rho_{n,mobile,0}(x) \text{ on } \Omega_n, \tag{44}$$

(ii) For the channel variable we only need to assign initial values for the states not directly connected to the compartments:

$$\pi_2(0) = \pi_{2,0}, \ldots, \pi_{m-1}(0) = \pi_{m-1,0}. \tag{45}$$

All initial conditions which are either functions or constant values are assumed to be non-negative. We would like to put this well-posed[1] mathematical model into a cell biology context. The boundary Γ_c is modelling the outer cell wall. The cell is embedded in a solution where the concentration of a ligand activating some receptor molecules can be kept controlled in a time interval $[0, t^*]$, with $0 > t^* > T$. After time t^* this ligand is washed away. The activated receptor activates a transcription factor (TF), and it is this activated transcription factor which is assumed to be our molecular species. The activation is close to the cell wall and described by the continuous function $c(t, y) > 0$. The TF cannot leave the cell after the activation of the receptors has stopped, equation (33). Now the TF is freely diffusing in the cytoplasm (equation (34)). Some TF molecules will hit the nuclear envelope modelled by Γ_n. In order to reach any gene inside the nucleus the TF has to cross a nuclear porous complex (NPC). In the current model the NPC is simply modelled by a linear channel, modelled by a linear Markov chain. It also has to be noted that the pores are not modelled as discrete entities. We assume there is a concentration of such channels across the membrane system, of course this is a mathematical abstraction perhaps not justified at a resolution of a confocal microscope. Inside the nucleus modelled by Ω_n the TF can possibly bind to some structure, for example the chromatin. The TF can also be released again. The density of the structure and therefore the probability of a TF molecule to bind or unbind is encoded in the functions $k_b(x) > 0$ and $k_d(x) > 0$. Finally there are regions in the nucleus where the TF is not able to enter, for example a nucleosome which is assumed to be too dense. Such regions are modelled by the domains Ω_1, Ω_2 and Ω_3. At time T we stop to look at this 'in-silico'[2] experiment.

3.1 Reactions among Several Species of Molecules

Our model proposed in the previous section, and composed of modules described earlier, is already having some complexity. Nevertheless we have to remind ourselves that in reality every such molecule will have to take part in different molecular reactions, mostly binding to form so-called complexes. A transcription factor for example will need to bind to another molecule[3] in order to be really able to cross the NPCs. To include reactions we will need to increase the number of molecular species. In case these additional species will be present in the same domains as the TF modelled before, each of these additional species

[1] This would not be too hard to prove by standard techniques. Nevertheless this is a non-standard PDE problem, mostly due to the interface dynamics defined on one of the boundaries.

[2] After implementation in a computer, which still needs a not straightforward discretisation step.

[3] For example to the RAN-GDP complex. See the RAN pathway.

will add the same number of state variables as was needed to model the TF. Reactions are transitions between species, and as those we can also formally define the mobile and immobile phase of the same species as two different populations of molecules. In this case equations (39) and (40) already show how reaction terms enter the model. We give a short overview to mass-action kinetics in a non-spatial setting. The resulting reaction terms would be needed to be incorporated to the spatially explicit model at each location x in a domain where such a reaction can take place. For simplicity we only consider and discuss here the well known deterministic mass-action reaction systems and fix the notation. It should be noted that the local dynamical system (i.e. for each location x in a compartment) derived from the reaction scheme in this interpretation is only valid with different implicit assumptions made for the mechanisms of the reactions and the properties and abundance of the different molecules involved. In particular there has been a law of large numbers being applied to the particle system, and the continuum limit is only represented by the so-called 'average dynamics', i.e. any stochastic fluctuations are assumed to be negligible. Also we assume each particle of every species remains unchanged in its properties during reactions and only 'varies' by forming molecules with other species or molecules of its own kind. Such a chemical mass-action reaction system with r reactions and m reacting species is then described by a time-continuous dynamical system defined for each $x \in \Omega$ which is directly associated to the reaction scheme. Each such reaction can be written in the form

$$\alpha_{1j}S_1 + \cdots + \alpha_{nj}S_n \xrightarrow{k_j} \beta_{1j}S_1 + \cdots + \beta_{nj}S_n, \quad j = 1, ..., r, \tag{46}$$

where the S_i, $1 \leq i \leq n$, are the chemical species and each $k_j > 0$ is the kinetic constant of the j-th reaction. The kinetic coefficients take into account all effects on the reaction rate apart from reactant concentrations, for example, temperature, light conditions, or ionic strength in the reaction. The coefficients α_{ij} and β_{ij} represent the number of S_i molecules participating in j-th reaction at reactant and product stages, respectively. The net amount of species S_i produced or consumed by the reaction is named the stoichiometric coefficient and defined by $n_{ij} := \beta_{ij} - \alpha_{ij}$. These coefficients are arranged in a *stoichiometric* matrix, denoted by N. The rate at which the j-th reaction takes place in mass-action kinetics takes the form of a monomial,

$$v_j(x, k_j) = k_j \prod_{i=1}^{m} (\rho^i)^{\kappa_{ij}},$$

where κ_{ij} is the molecularity of the species S_i in the j-th reaction, and ρ^i is the concentration of the ith species. Also in our mass-action kinetic interpretation the kinetic exponent κ_{ij} reduces to being simply α_{ij}. Kinetic exponents are arranged in a *kinetic* matrix, denoted by κ. The time evolution of the species concentrations is described by the following initial value problem:

$$\dot{\rho} = Nv(\rho, k), \tag{47}$$

$$\rho(0) > 0, \tag{48}$$

where $\rho(0)$ are initial molecular concentrations. The vector

$$\rho(t,x) = (\rho^1(t,x), \ldots, \rho^n(t,x))^T$$

is describing the concentrations of the n different molecular species $S_1, \ldots, S_n$ at *any* location x in one of the compartments of our membrane system. Equation (47) is delivering the so called reaction terms, it does neglect all transport or transportation over membranes of any of the species. We need to enlarge the state space associated to our complete model for a single molecular species as described in section 3. The state of the extended system is now given by the molecular concentrations

$$\rho_c(t,x) = (\rho_c^1(t,x), \ldots, \rho_c^n(t,x))^T, \rho_n(t,x) = (\rho_n^1(t,x), \ldots, \rho_n^n(t,x))^T,$$

and the states $\pi(t,y) = (\pi_1^1(t,y), \ldots, \pi_m^n(t,y))$ which can now be interpreted as a $m \times n$-matrix. Of course we make a number of assumption implicitly, like the one that molecules of different species do not interact in the channel etc. All such issues would need to be fine-tuned and checked in a realistic modelling attempt. We should also note that each single equation of the model in the beginning of section 3 is now becoming a system of n equations corresponding to the n different species we have introduced. Finally interesting qualitative behaviours such as bistability and oscillations have been observed in such reaction systems of mass-action type, see [1,6,9,17,11]. They can be interpreted to result from a bifurcation, i.e. a qualitative change in the behaviour of the system's solutions when one or more of the parameters are varied. Hence, a common approach in identifying such behaviour has been to derive conditions under which the system is able to undergo an associated bifurcation, see [29,11]. Such considerations hold for each location in space independently. In combination with transport of molecules we can expect that from local complex dynamics (such as oscillations) very complex spatial *pattern formation* can arise. See for example different chapters in [24], especially chapter 1 by Fiedler and Scheel, and chapter 3 by Paul Fife. Very often elementary chemical reactions of mass-action type are too complex for modelling biological systems. After time scaling the reactions schemes can be often simplified and then are called 'enzyme kinetics', see among others [40,23].

3.2 The Necessary Discretisation

There a various way to discretise the PDE models we have finally derived. It should be noted that the PDE did result from up-scaling of discrete objects, for example particles jumping on a fine grid which grid size was scaled to zero. In other words, and for later discussion, we make a step from discrete stochastic processes to continuum models, and finally a step back to discrete models which are discretisations of continuum models, as those are the only ones that can be currently interpreted by present standard computer architecture. We choose a form of a so-called mesh-free discretisation based on a partition of unity (PUM), a class of generalised finite element methods. We explain the discretisation step for a single given diffusion equation in one of our compartments. Let H be an

appropriate Hilbert space ($H = H^1$ for 2nd order problems like the diffusion equation). We can obtain the variational form of the PDE using a continuous bilinear form $a : H \times H \to R$ and a linear form $l \in H'$ along with appropriate boundary or interface conditions. The final problem we seek to solve may be summarised as

$$\text{Find} \quad u \in H \quad \text{s.t.} \quad a(u, v) = l(v) \qquad \forall v \in H. \tag{49}$$

The basic method of discretisation in the PUM framework is then given by the following steps:

- Given a domain Ω on which a linear scalar PDE is defined, open sets called *patches* are used to form a cover of the domain. ($\Omega_N := \{\omega_i\}_{i=1}^N$, with $\overline{\Omega} \subset \bigcup_i \omega_i$).
- A partition of unity $\{\varphi_i\}_{i=1}^N$ subordinate to the cover is constructed.
- The local function space on patch ω_i, $1 \leq i \leq N$, is given by $\mathcal{V}_i := \text{span}\{\psi_i^k\}_{k=1}^{p_i}$, with $\{\psi_i^k\}_{k=1}^{p_i}$ being a set of base functions defining the approximation space for each patch. The global approximation space, also called the trial or the *PUM space*, is defined by $V_{\text{PU}} := \text{span}\{\varphi_i \psi_i^k\}_{i,k}$. Replacing H by the finite dimensional subspace[4] V_{PU}, a global approximation u_h to the unknown solution u of the PDE is defined as a (weighted) sum of local approximation functions on the patches:

$$u_h(x) = \sum_{i=1}^{N} \varphi_i(x) \left(\sum_k \xi_i^k \psi_i^k(x) \right).$$

- The unknown coefficients ξ_i^k are determined by substituting the above approximation into the PDE and using the method of weighted residuals to derive an algebraic system of equations

$$A\xi = b. \tag{50}$$

More detail on the PUM, including a description of its approximation properties and how to construct the PUM space, may be found in, for example, [2,3,4,39]. We have implemented the PUM in a C++ code called the *Generic Discretisation Framework* (*GDF*). See also [12,13,14].

4 What Can Be Measured?

Once the model is implemented on a computer we can compare the simulated solution with a measured concentration of molecules in a given cell. Typically this would currently be best done in an in-vivo situation with the help of fluorescent markers with which the molecules under investigation have been tagged. As

[4] Note that V_{PU} is conforming for the Neumann problems we are concerned with in this article.

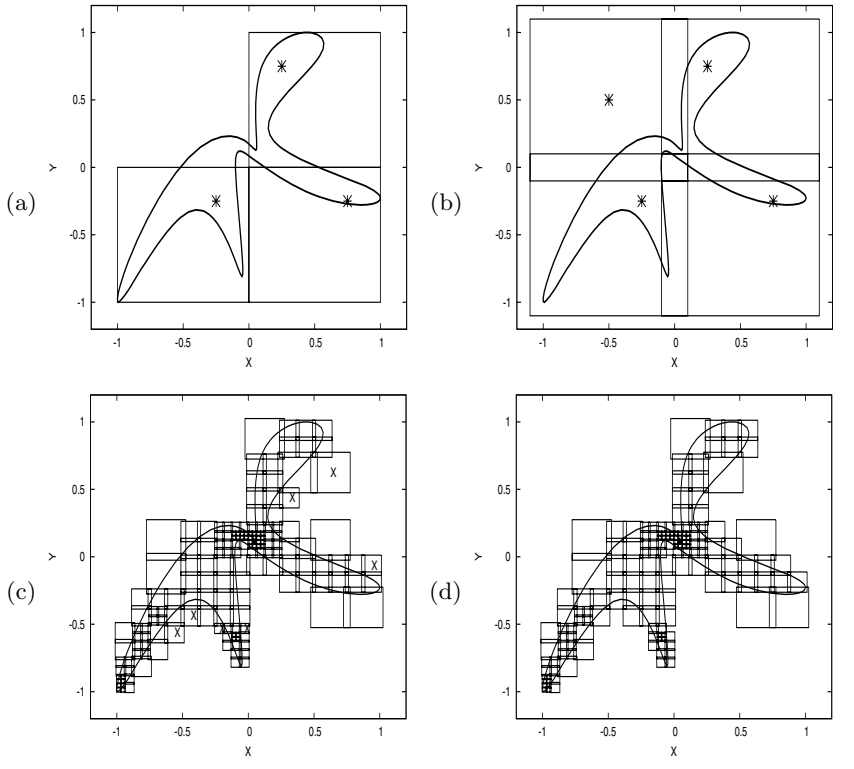


Fig. 4. Key stages of cover construction for a complex shaped compartment. **(a)** Three points distributed randomly in a complex domain and the initial cover. **(b)** An increase in the number of patches so that the whole domain is covered. Patches have also been extended by $\alpha = 1.2$. **(c)** Optional refinements of the cover. For clarity, the points are omitted and only the associated patch pictured. Seven patches whose intersection with the domain are subsets of another patch are labelled 'X' and will be removed in the final stage of basic cover construction. **(d)** The final cover of 159 patches with the seven patches from frame (c) removed.

there are only very few colours available[5] the number of different species that can be observed at the same time is very limited, and most of the time is just a single species. We observe the distribution of the molecules with the help of a microscope, in this context this will be a confocal microscope most of the time. This can be done at different magnifications which introduces the spatial scale at which we have to consider the problem. Interestingly enough the pictures we will retrieve from the microscope are again discrete, i.e. pixel based. This means we are comparing a discrete solution of a continuum model with the discretised image of a (on the typical scales given by the resolution chosen) continuous

[5] The first one introduced and the one best known is GFP, for Green Fluorescent Protein.

distribution of fluorescence in the sample cell. The normal situation we will encounter is that the fluorescence distribution of the tagged molecules when starting the observation will be static, i.e. in equilibrium. This is because most of the experimental conditions start from an equilibrium. If there is for example a diffusion process, the molecules will have had enough time to evenly distribute in all compartments which they can enter. In order to make processes visible we will have to perturb the system. This can be done with the help of a laser with which we can bleach the fluorescent molecules. In other words we can implement a sink term for fluorescence[6] and the result is a dynamic situation where all the mechanisms responsible for protein distribution are now acting together to change the measured fluorescence distribution over time. The most frequently used approach is FRAP (Fluorescence Recovery After Photobleaching) where a certain part of the cell is bleached, i.e. all fluorescent molecules in a given area are bleached in a very short period of time. Subsequently it can be measured how the now bleached area is recovering its fluorescence. This can only happen if the tagged molecules outside the bleach area can move, for example again by a simple diffusion process. We frequently use also longer bleaching periods. This can be helpful, for example in understanding how many fluorescent molecules are estimated to be in a closed compartment, or by equilibrating a flux into a compartment with the sink created by the laser beam. The literature on FRAP and the other bleaching techniques is huge and we do not try here to give a complete overview. A paper describing and applying techniques very close to techniques we frequently use in the laboratory is [31].

The are many more ways of measuring specific parameters and mechanisms of the model proposed, most of them related to in-vitro measurements, for example measuring kinetic constants etc. We cannot go into any detail here, but emphasise that information from different sources (and on different scales) will be needed in a successful modelling attempt. Geometrical information is discussed in the next subsection.

4.1 Cell Compartmental Geometry

The images of Fig. 5 did not allow to retrieve the geometry of the chloroplast. The membrane system inside a chloropast is very complex, consisting of various so called thylakoid staples which host the light harvesting complexes. In this case it would be better get the problem geometry from other sources, for example from electron microscopy (EM) with a much higher resolution. Sometimes however the confocal microscopy resolution is sufficient. Figure 6 shows a fibroplast cell which is almost flat[7]. Here the essential geometry of the membrane system (plasma membrane, nuclear envelope) could be recovered.

[6] There are also activatable fluorescent molecules which can be activated by a laser beam. In this case the sink becomes a source.

[7] This justifies a 2D approach in the modelling.

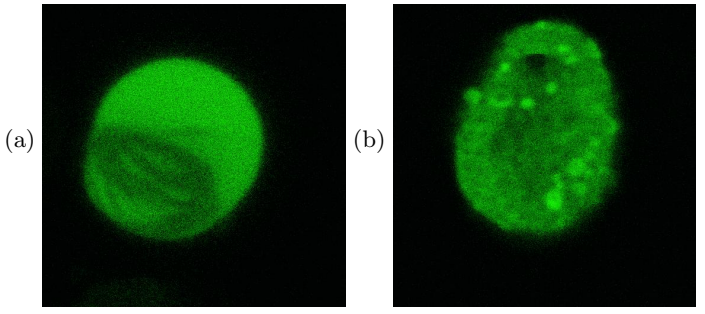


Fig. 5. The start of two different continuous bleaching experiments with the help of a confocal microscope. Courtesy of Colin Robinson's laboratory (University of Warwick). Each picture shows a single chloroplast inside a single pea protoplast (the outer rigid cell wall has been removed) plant cell. **(a)** A bleaching example where pure GFP was translocated into the chloroplast. This molecule cannot bind to any membranes inside the chloroplast. As this molecule is rapidly diffusing the bleaching area is only visible indirectly as a larger area of fluorescence depletion in the lower part of the chloroplast. **(b)** A membrane protein has been translocated into the chloroplast. The bleaching area in the top of the picture can be clearly seen as the protein is largely immobile, most likely bound to the membrane system. Illustration taken from [13].

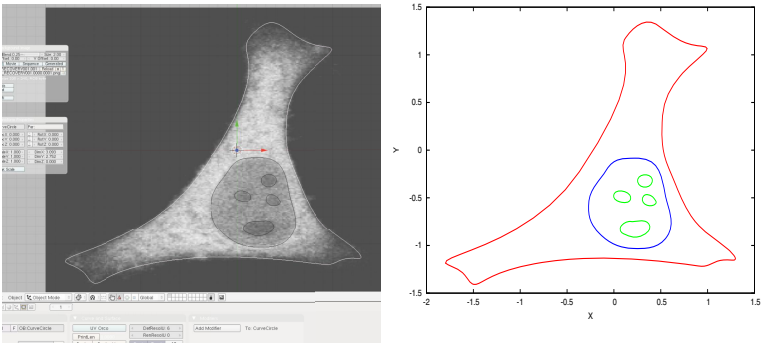


Fig. 6. Domain with nested subdomains. Left the original fibroplast microscopy image inside a modelling programme to retrieve the (sub-)domain shapes, courtesy of Gimmi Ratto, see also [33]. Right: The main domain represents the cell body, the main subdomain represents the nucleus, and the four inner sub-subdomains (green) represent nucleoli. Illustration taken from [13].

4.2 Simulation Results

In order to compare measurements with simulations we best visualise the computed solutions on the geometry we have retrieved from the experimental situation. We illustrate this in Figure 7 with the help of the fibroblast example.

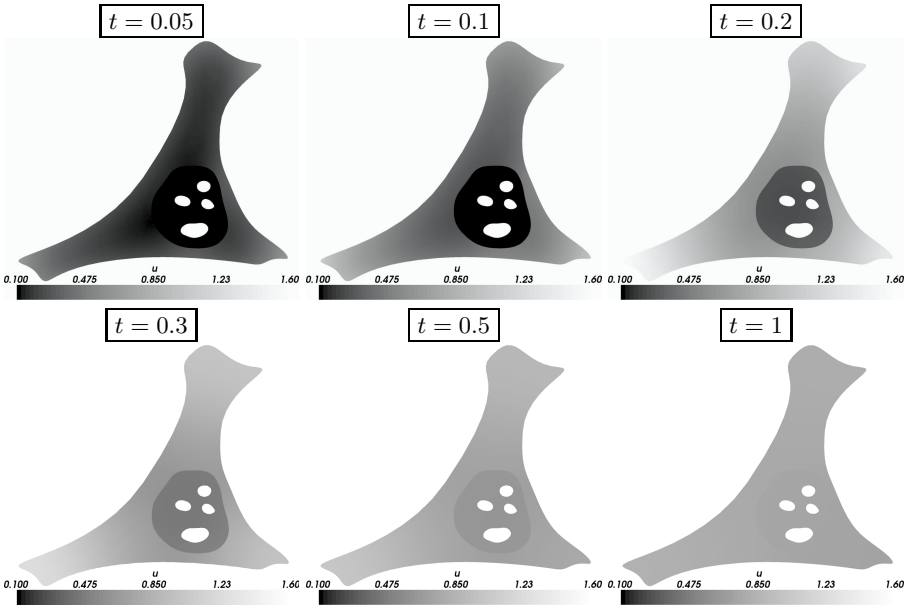


Fig. 7. Concentration function $\rho(t, x)$ at selected times for a simulation on the entire two-dimensional fibroblast shape. In this simplified signalling problem a molecule is released (activated) uniformly at the plasma membrane, diffuses inside the cytoplasm, can enter the nucleus (a sub-domain, i.e. a nested domain of level 1) following a simple linear relationship based on concentration differences (not modelling the NPCs in detail), and finally distributes itself inside the nucleus where it can get bound to the nucleoli modelled as sub-sub-domains (nested domains of level 2). Illustration taken from [13].

5 The Cell as an Information Processing Device

Membrane computing and modelling and simulation of cellular molecular distributions have been defined and introduced for completely different purposes. Membrane computing is very generally speaking taking up successfully ideas from biology to theoretically analyse algorithms. The concept is to structure algorithms that eventually can solve certain computational tasks in finite time. This is a step away from the Turing machine. This theoretical device is as unstructured as possible, i.e., it was designed to investigate a very general class of algorithms and checks whether a given programme encoded in bits stops after performing only finitely many discrete steps (stopping problem). Algorithms solving problems in a modern world, like regulating traffic in a city, are surely necessarily much more structured. They are expressed in languages that are object oriented, which means there are surely 'membranes' around the objects that protect local variables to be overwritten etc. Nevertheless information has to enter and leave the objects. This is indeed very close to the biological situation of a cell. As mentioned in the introduction it is very likely that the complex membrane systems we see in eukaryotic cell are linked to the fact that cells in a

multi-cellular organisation have to solve much more different tasks. Such tasks are always better and more robustly solved if they can be distributed to different specialised sub-units, the organelles. The cell developed different mechanisms how molecules can pass the membranes, in cell biology called 'translocation'. It was on purpose we have chosen a transcription factor (TF) to explain how molecular processes do solve the task of reacting to signals stemming from outside the cell. Such processes are usually called 'signalling pathways'. The transcription factor is for example released or better activated close to the plasma membrane where the receptors measuring the outside signal[8] are located. They finally have to reach the respective gene which needs to be activated in order to respond to the changes of the environment, here signalled by a changing signalling extracellular molecular concentration. Nevertheless the task might not be so straightforward. The internal state of the cell may act as a filter of the incoming signal. A gene might only be switched on if a transcription factor reaches the gene in the right way. It may be required that the signal is repeated just with the right amplitude and frequency[9]. To work as a filtering information processing device the cell needs to be able to adapt its internal state, including the regulation of membrane proteins enabling other molecules to cross membranes. To understand these processes it was always good practise to try a forward simulation, i.e. to model the different sub-processes that might lead to the observed molecular distributions, assemble them in a system, and compare prediction and measurement for a given period of time. In this paper we have defined a possible framework with which such computations can be performed. It is striking how close the concepts are in relation to ideas from membrane computing. This is no coincidence, as both approaches need to directly abstract cellular biological performance.

6 Discussion

We can easily see the similarities and differences between membrane computing and cellular interface problems (CIP) when we look at the definitions of both 'processes'[10]. Following the definition of a transition P system in [32] (of degree $m \geq 1$) such a system is a construct of the form

$$\Pi = (O, C, \mu, w_1, w_2, \ldots, w_m, R_1, R_2, \ldots, R_m, i_o),$$

with

1. O is the (finite and nonempty) alphabet of objects,
2. $C \subset O$ is the set of catalysts,

[8] For example a hormone.

[9] For example such processes are important for synaptic plasticity where only a repeated signal should be interpreted as 'learning', i.e. increasing synaptic transmission strength.

[10] Both membrane computing and cellular interface problems can be called processes as they necessarily change both generically their state in time.

3. μ is a membrane structure, consisting of m membranes, labeled $1, 2, \ldots, m$; we say that the membrane structure, and hence the system, is of degree m,
4. $w_1, w_2, \ldots, w_m$ are strings over O representing the multisets of objects present in regions $1, 2, \ldots, m$ of the membrane structure,
5. $R_1, R_2, \ldots, R_m$ are nite sets of evolution rules associated with regions $1, 2, \ldots, m$ of the membrane structure,
6. i_o is either one of the labels $1, 2, \ldots, m$, and the respective region is the output region of the system, or it is 0, and the result of a computation is collected in the environment of the system.

The rules related to this system are of the form $u \to v$ or $u \to v\delta$, with $u \in O^+$ and $v \in (O \times Tar)^*$, where $Tar = \{here, in, out\}$. The rules are applied to be maximally parallel. We first focus on similarities. The membrane structure μ can be interpreted as identical in the cellular interface problem, but there will be in addition the geometrical information needed for the simulation. The 'molecules' in the P systems are also represented by a finite number of different species, labelled $w_1, w_2, \ldots$. There is no direct relationship between the evolution rules $R_1, R_2, \ldots, R_m$ and the transitions in the CIP. Of course the mass-action kinetic reactions we have briefly discussed do deliver such rules, but there are other transition rates related to the transport process (not incorporated to the transition P system) and the translocation processes[11] over membranes. The labelling of the species in order to follow their membership to certain compartments is done in the same way as in the transition P system.

Many differences between P system and CIP result from the fact that the CIP is formulated completely discrete, whereas the CIP on its macroscopic level is completely formulated in a continuous framework. This can be seen by comparing the alphabet O of the P system and the vector of concentrations ρ formulated for the CIP. This can be explained by introducing the concept of scale, both in space and time. For a P system temporal scale is largely irrelevant as it is a powerful computational concept. Like for a Turing machine it is important to perform the steps defined by the rules of the P system and the definition of a universal clock where discrete steps (events) are performed in a maximally parallel way as long as the system has not reached its final configuration. For the CIP scale is crucial, both for time and space. Processes can take place at different temporal and spatial scales, and during the so called multi-scale analysis (see [38,34]) these processes will look different in the final outcome of the model. To make this point was the reason to include a long discussion of the channel dynamics where we motivated how to derive the corresponding interface conditions.

It could be argued that this distinction between discrete and continuous frameworks is artificial. In fact we could just define all transitions of the CIP on a microscopic scale, where discrete particles would follow stochastic processes. This is in fact a true statement. Nevertheless a scale problem would arise also in this framework, the events of different molecular processes would happen at different time scales. This would mean we would need to go to the smallest time scale present in the system, and define all other processes in terms of this basic scale.

[11] Just as an example we have discussed in detail a given channel dynamics.

The result would be an infinitely complex system which we presently could not handle even on large computers. The reason for this is that we would need to give up using all different averaging procedures which have been developped to derive simpler, macroscopic, often called 'effective' equations on relatively large temporal and spatial scales. It is in fact 'averaging' in a wider sense that motivates the use of continuum models if direct modelling and simulation of (biological) temporal and spatial processes is the given task. It would be interesting to investigate if such concepts also could apply for the further development of P systems.

References

1. Aguda, B.D., Clarke, B.L.: Bistability in chemical reaction networks: theory and application to the peroxidase-oxidase reaction. J. Chem. Phys. 87, 3461–3470 (1987)
2. Babuška, I., Banerjee, U., Osborn, J.E.: Meshless and generalized finite element methods: a survey of some major results. In: [26], p. 120. Springer, Berlin (2003)
3. Babuška, I., Melenk, J.M.: The partition of unity method. Internat. J. Numer. Methods Engrg. 40(4), 727–758 (1997)
4. Belytschko, T., Krongauz, Y., Organ, D., Fleming, M., Crysl, P.: Meshless methods: An overview and recent developments. Computational Methods in Applied Mechanical Engineering 139, 3–47 (1996)
5. Bronnikova, T.V., Fed'kina, V.R., Schaffer, W.M., Olsen, L.F.: Period-doubling bifurcations in a detailed model of the peroxidase-oxidase reaction. J. Phys. Chem. 99, 9309–9312 (1995)
6. Clarke, B.L.: Stability of complex reaction networks. In: Prigogine, I., Rice, S. (eds.) Advan. Chem. Phys., vol. 43, pp. 1–216. Wiley, New York (1980)
7. Clarke, B.L., Jiang, W.: Method for deriving Hopf and saddle-node bifurcation hypersurfaces and application to a model of the Belousov-Zhabotinskii reaction. J. Chem. Phys. 99, 4464–4476 (1993)
8. Cornish-Bowden, A., Hofmeyer, J.-H.S.: The role of stoichiometric analysis in studies of metabolism: an example. J. Theor. Biol. 216, 179–191 (2002)
9. Craciun, G., Tang, Y., Feinberg, M.: Understanding bistability in complex enzyme-driven reaction networks. PNAS 30(103), 8697–8702 (2006)
10. Domijan, M., Kirkilionis, M.: Graph Theory and Qualitative Analysis of Reaction Networks. Networks and Heterogeneous Media 3, 95–322 (2008)
11. Domijan, M., Kirkilionis, M.: Bistability and Oscillations in Chemical Reaction Systems. Journal of Mathematical Biology (in press, 2008)
12. Eigel, M., George, E., Kirkilionis, M.: A Meshfree Partition of Unity Method for Diffusion Equations on Complex Domains. IMA Journal of Numerical Analysis (in press, 2008)
13. Eigel, M., Erwin, G., Kirkilionis, M.: The Partition of Unity Meshfree Method for Solving Transport-Reaction Equations on Complex Domains: Implementation and Applications in the Life Sciences. In: Griebel, M., Schweitzer, A. (eds.) Meshfree Methods for Partial Differential Equations IV. Lecture Notes in Computational Science and Engineering, vol. 65. Springer, Heidelberg (2009)
14. Eigel, M.: An adaptive mashfree method for reaction-diffusion processes on complex and nested domains. PhD thesis, University of Warwick (2008)

15. Field, R.J., Körös, E., Noyes, R.M.: Oscillations in chemical systems. 2. Thorough analysis of temporal oscillation in bromate-cerium-malonic acid system. J. Am. Chem. Soc. 94(25), 8649–8664 (1972)
16. Ferry, J.G., House, C.H.: The Stepwise Evolution of Early life Driven by Energy Conservation. Molecular Biology and Evolution 23(6), 1286–1292 (2006)
17. Field, R.J., Noyes, R.M.: Oscillations in chemical systems. IV. Limit cycle behavior in a model of a real chemical reaction. J. Chem. Phys. 60, 1877–1884 (1974)
18. Goldbeter, A., Dupont, G.: Allosteric regulation, cooperativity, and biochemical oscillations. Biophys. Chem. 37, 341–353 (1990)
19. Guckenheimer, J., Holmes, J.P.: Nonlinear Oscillations, Dynamical Systems and Bifurcations of Vector Fields. Applied Mathematics Sciences, vol. 42. Springer, Heidelberg (2002)
20. Heinrich, R., Schuster, S.: The Regulation of Cellular Processes. Chapman & Hall, Boca Raton (1996)
21. Hunt, K.L.C., Hunt, P.M., Ross, J.: Nonlinear Dynamics and Thermodynamics of Chemical Reactions Far From Equilibrium. Annu. Rev. Phys. Chem. 41, 409–439 (1990)
22. Ivanova, A.N.: Conditions for uniqueness of stationary state of kinetic systems, related, to structural scheme of reactions. Kinet. Katal. 20(4), 1019–1023 (1979)
23. Keener, J., Sneyd, J.: Mathematical Physiology. Springer, Heidelberg (1998)
24. Kirkilionis, M., et al. (eds.): Trends in Nonlinear Analysis. Springer, Heidelberg (2003)
25. Kirkilionis, M.: Reaction systems, graph theory and dynamical networks. In: Gauges, R., et al. (eds.) 5th Workshop on Computation of Biochemical Pathways and Genetic Networks, pp. 131–150. Logos-Verlag (2008)
26. Kirkilionis, M., Sbano, L.: An Averaging Principle for Combined Interaction Graphs. Part I: Connectivity and Applications to Genetic Switches. In: Advances in Complex Systems (2008); in revision. Also available as WMI Preprint 5/2008
27. Klonowski, W.: Simplifying principles for chemical and enzyme reaction kinetics. Biophys. Chem. 18, 73–87 (1983)
28. Krischer, K., Eiswirth, M., Ertl, G.: Oscillatory CO oxidation on Pt(110): Modeling of temporal self-organisation. J. Chem. Phys. 96, 9161–9172 (1992)
29. Kuznetsov, Y.: Elements of Applied Bifurcation Theory, 2nd edn. Applied Mathematical Sciences, p. 112. Springer, Heidelberg (1998)
30. Melenk, J.M., Babuška, I.: The partition of unity finite element method: basic theory and applications. Comput. Methods Appl. Mech. Engrg. 139(1-4), 289–314 (1996)
31. Misteli, T., Gunjan, A., Hock, R., Bustink, M., David, T.: Dynamic binding of histone H1 to chromatin in living cells. Nature 408, 877–881 (2000)
32. Paun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
33. Ratto, G.M., Pizzorusso, T.: A kinase with a vision: Role of ERK in the synaptic plasticity of the visual cortex. Adv. Exp. Med. Biol. 557, 122–132 (2006)
34. Pavliotis, G.A., Stuart, A.M.: An Introduction to Multiscale Methods. Springer, Heidelberg (2008)
35. Perelson, A.S., Wallwork, D.: The arbitrary dynamic behavior of open chemical reaction systems. J. Chem. Phys. 66, 4390–4394 (1977)
36. Sbano, L., Kirkilionis, M.: Molecular Reactions Described as Infinite and Finite State Systems. Part I: Continuum Approximation. Warwick Preprint 05/2007
37. Sbano, L., Kirkilionis, M.: Molecular Reactions Described as Infinite and Finite State Systems Part Ii: Deterministic Dynamics and Examples. Warwick Preprint 07/2007

38. Sbano, L., Kirkilionis, M.: Multiscale Analysis of Reaction Networks. Theory in Biosciences 127, 107–123 (2008)
39. Schweitzer, M.A.: Efficient implementation and parallelization of meshfree and particle methods—the parallel multilevel partition of unity method, pp. 195–262. Springer, Berlin (2005)
40. Siegel, I.H.: Enzyme Kinetics. Wiley, Chichester (1975)
41. Selkov, E.E.: Self-oscillations in glycolysis. 1. A simple kinetic model. Eur. J. Biochem. 4, 79–86 (1968)
42. Slepchenko, B.M., Terasaki, M.: Cyclin aggregation and robustness of bio-switching. Mol. Biol. Cell. 14, 4695–4706 (2003)
43. Tyson, J.J., Chen, K., Novak, B.: Network dynamics and cell physiology. Nat. Rev. Mol. Cell Biol. 2, 908–916 (2001)

A Multiscale Modeling Framework Based on P Systems

Francisco José Romero-Campero[1], Jamie Twycross[1,2], Hongqing Cao[1], Jonathan Blakes[1], and Natalio Krasnogor[1]

[1] Automated Scheduling, Optimisation and Planning Research Group
School of Computer Science, Jubilee Campus, University of Nottingham
Nottingham NG8 1BB, United Kingdom
[2] Centre for Plant Integrative Biology
Sutton Bonington Campus, University of Nottingham
Nottingham LE12 5RD, United Kingdom
{fxc,jpt,hxc,jvb,nxk}@cs.nott.ac.uk

Abstract. Cellular systems present a highly complex organization at different scales including the molecular, cellular and colony levels. The complexity at each one of these levels is tightly interrelated. Integrative systems biology aims to obtain a deeper understanding of cellular systems by focusing on the systemic and systematic integration of the different levels of organization in cellular systems.

The different approaches in cellular modeling 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 modeling 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, compartmentalization is described using membrane structures and finally cellular colonies and tissues are modeled 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 characterize specific cellular processes. One of our current research lines focuses on the design of cell systems biology models exhibiting a prefixed behavior through the automatic assembly of these cellular modules. Our approach is equally applicable to synthetic as well as systems biology.

1 Introduction

Models in systems biology has been recently classified according to their semantics into *denotational* and *operational* models [6]. Models with *denotational semantics* are the classical approach in modeling cellular systems which uses a

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 63–77, 2009.

set of equations to describe how the quantities of the different molecular species are related to each other over time. The classical example are ordinary and partial differential equations. In this case the behavior of the system is obtained by approximating numerically these equations. On the other hand, *computational models* have *operational semantics* which describe the behavior of the system using an algorithm or list of instructions that can be executed by an abstract machine. The models developed within this last framework has been termed recently *executable biology* [6]. In this case a more detailed description of the processes producing the behavior of the system is provided.

Several formal computational approaches have been proposed to model cellular systems like Petri nets [10] and process algebra [19]. They mainly focus on system specification at the molecular level: membranes, compartmentalization and cellular colonies are seldom described. This fact makes it difficult to study multicellular systems whose function is determined by molecular interactions.

Membrane computing is a branch of natural computing inspired directly from the structure and functioning of the living cell [14]. It has been applied to cellular modeling as one of the few computational frameworks which presents an integrative approach to multiscale systems ranging from the molecular to the multicellular level. Specifically, it represents the molecular interaction level of living cells using objects or strings and rewriting rules; the compartmental/cellular level using membranes; and the colony level using collections of membranes called membrane structures. The devices of this computational paradigm are referred to as *P systems*. Although most research in P systems focuses on the study of the computational power of the different proposed variants, recently their application as a modeling formalism to cellular systems is emerging [3,4,7,11,17,20,21,15]. In this paper we discuss through a running example the use of P systems as a multiscale modeling framework for cell systems biology models.

The paper is organized as follows. Stochastic P systems for cellular modeling are introduced in Section 2. Section 3 presents the running example used throughout this paper. The modeling principles in P systems are described in Section 4. Modularization in P systems is briefly discussed in Section 5. Finally, conclusions and future work are discussed in Section 6.

2 Stochastic P Systems

The original strategy for the application of the rewriting rules in P systems was based on maximal parallelism and non-determinism [13]. This strategy does not represent the rate at which molecular interactions take place as every object that can evolve according to any rule must evolve in a single computation step, without taking into account that some molecular interactions are more frequent than others. Moreover, the real time evolution of cellular systems is not captured as all the computation steps are assumed to be of the same time length, neglecting the fact that some molecular interactions are faster than others.

Different strategies for the application of the rewriting rules in P systems have been studied [5,8]. Specifically, a sequential stochastic strategy based on *Gillespie's theory of stochastic kinetics* [9] was introduced in order to overcome the two

previous problems when developing a modeling framework for cellular systems biology based on P systems [16]. Here we refer to this variant as *stochastic P systems.*

Definition 1 (Stochastic P Systems). *A Stochastic P system is a construct:*

$$\Pi = ((\Sigma_{obj}, \Sigma_{str}), L, \mu, M_{l_1}, \ldots, M_{l_m}, (R_{l_1}^{obj}, R_{l_1}^{str}), \ldots, (R_{l_m}^{obj}, R_{l_m}^{str})),$$

where:

- Σ_{obj} *is a finite alphabet of objects representing molecular species whose internal structure is not relevant in the functioning of the system under study.*
- Σ_{str} *is a finite alphabet of objects representing relevant parts of some molecular species in the system. These objects are arranged into strings describing the structure of molecular species.*
- $L = \{l_1, \ldots, l_m\}$ *is a finite alphabet of symbols representing compartment labels used to identify compartment classes. Compartments with the same label share the same class, i.e., set of rewriting rules and initial multisets.*
- μ *is a membrane structure consisting of* $n \geq 1$ *membranes defining compartments identified in a one to one manner with values from* $\{1, \ldots, n\}$ *and labeled with elements from* L.
- $M_{l_t} = (w_t, s_t)$, *for each* $1 \leq t \leq m$, *is the initial state of the compartments from the class identified by label* l_t, *where* $w_t \in \Sigma_{obj}^*$ *is a finite multiset of individual objects and* s_t *is a finite set of strings over* Σ_{str}. *A multiset of objects, obj is represented as* $obj = o_1 + o_2 + \ldots + o_p$ *with* $o_1, \ldots, o_p \in \Sigma_{obj}$. *Strings are represented as follows* $\langle s_1 \cdot s_2 \cdots s_q \rangle$ *where* $s_1, \ldots, s_q \in \Sigma_{str}$.
- $R_{l_t}^{obj} = \{r_1^{obj,l_t}, \ldots, r_{k_{obj,l_t}}^{obj,l_t}\}$, *for each* $1 \leq t \leq m$, *is a finite multiset of rewriting rules on multisets of objects associated with compartments of the type specified by the label* l_t. *The rewriting rules on multisets of objects are of the following form:*

$$r_j^{obj,l_t} : \; obj_1 \, [\, obj_2 \,]_l \overset{c_j^{obj,l_t}}{\longrightarrow} obj_1' \, [\, obj_2' \,]_l \tag{1}$$

with $obj_1, obj_2, obj_1', obj_2'$ *some finite multisets of objects from* Σ_{obj} *and* l *a label from* L. *These rules are multiset rewriting rules that operate on both sides of membranes, that is, a multiset* obj_1 *placed outside a membrane labeled by* l *and a multiset* obj_2 *placed inside the same membrane can be simultaneously replaced with a multiset* obj_1' *and a multiset* obj_2', *respectively.*

Note that a constant c_j^{obj,l_t} *is associated specifically with each rule. This constant will be referred to as stochastic constant and is key to provide P systems with a stochastic extension as it will be used to compute the probability and time needed to apply each rule. This constant depends only on the physical properties of the molecules and compartments involved in the reaction described by the rule like temperature, pressure, pH, volume, etc.*

- $R_{l_t}^{str} = \{r_1^{str,l_t}, \ldots, r_{k_{str,l_t}}^{str,l_t}\}$, *for each* $1 \leq t \leq m$, *is a finite set of rewriting rules on multisets of strings and objects associated with compartments of the type defined by* l_t *and of the following form:*

$$r_j^{str,l_t} : \; [\, obj + str \,]_l \overset{c_j^{str,l_t}}{\longrightarrow} [\, obj' + str'; str_1' + \ldots + str_s' \,]_l \tag{2}$$

with obj, obj' multisets of objects over Σ_{obj} and $str, str', str'_1, \ldots, str'_s$ strings over Σ_{str}. These rules operate on both multisets of objects and strings. The objects obj are replaced by the objects obj'. Simultaneously a substring str is replaced by str' whereas the strings $str'_1, \ldots, str'_s$ are produced to form part of the content of the compartment. In the same way as for rewriting rules on multisets of objects a stochastic constant c_j^{str,l_t} is associated with each rule.

The previous definition is provided with a stochastic strategy for the application of the rewriting rules by extending the Gillespie algorithm to the multicompartmental structure of P systems. The resulting algorithm has been referred to as the *Multicompartmental Gillespie Algorithm* (MGA) [16]. The Gillespie algorithm [9] can only be applied directly in a single, fixed and well mixed volume. In our approach the first step consists of treating each compartment defined by a membrane as a fixed and well mixed volume where the rewriting rule to be applied and the elapsed time before its application is computed using the Gillespie Direct Method. Our algorithm then applies the corresponding rules following the order determined by these waiting times. After the application of each rule the algorithm recomputes the rules to be applied and the waiting times in the compartments affected by the application of the last rule using the Gillespie Direct Method. Finally, the MGA halts when a prefixed simulation time is reached or no further rules can be applied.

3 Running Example

In order to illustrate our modeling framework we will use an abstract gene regulation system inspired from the functioning and structure of the *lac operon* in *Escherichia coli* (*E. coli*). This operon consists of three structural genes, *lacZ*, *lacY* and *lacA*, located sequentially on the genome and transcribed into one single mRNA. Their protein products are involved in the sensing, uptake and metabolism of lactose. The transcription of the *lac operon* is both positively and negatively regulated and it is considered a canonical example of gene transcription regulation in prokaryotes [18].

The linear structure of the *lac operon* (Figure 1) starts with a region called *cap* where the activator protein CRP binds and increases the rate of transcription. Following this site there is an operator sequence that we will refer to as *op* where the repressor protein LacI binds to stop transcription. The structural genes *lacZ*, *lacY* and *lacA* then follow. The first gene *lacZ* codifies the enzyme β-galactosidase involved in the metabolism of lactose by cleaving it into glucose and galactose; allolactose appears as a byproduct of this reaction. The protein product of the second gene *lacY* is a permease that associates to the cell membrane and acts as a pump transporting lactose into the cell. The function of the protein coded in the last gene *lacA* is not yet fully understood.

The regulation of the *lac operon* allows *E. coli* to express the genes in the operon only when it is more beneficial for the cell. In the absence of lactose in the media the repressor LacI binds to the operator *op* preventing the structural

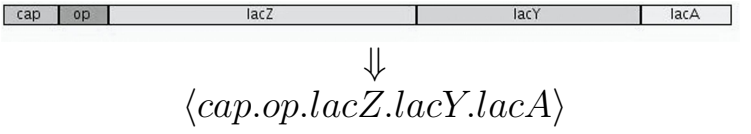


Fig. 1. A schematic representation of the structure of the *lactose operon* (top) and its representation as a string (bottom)

genes from being transcribed since they are not needed under these conditions. Nevertheless, occasionally the repressor drops from the operator producing a basal transcription of the operon.

When lactose becomes available it starts to be transported inside the cell by the basal number of LacY proteins on the cell surface. Once in the cytoplasm it interacts with the basal number of β-galactosidase producing as a byproduct *allolactose*. Allolactose in turn binds to the repressor LacI and changes its state so it cannot bind to the operator allowing transcription of the structural genes. The resulting increase in production of LacY and β-galactosidase forms a positive feed-back loop increasing the number of allolactose molecules which interact with the repressors preventing premature termination of transcription.

The *lac operon* is also under positive regulation by the protein CRP. This protein is activated by the glucose transport system and when active it binds to the *cap* site facilitating transcription. Even in the presence of lactose if glucose is present in the media CRP will not be active as the transport system will be occupied, pumping glucose into the cell. Therefore CRP will not bind to the operon to assist transcription. Only in the presence of lactose and absence of glucose will CRP be active and bound to the operon, producing the full transcription of the operon.

This gene regulation system will be used in the following section as the running example illustrating our modeling principles.

4 Modeling Principles

The complexity of cellular systems is organized into different levels ranging from the molecular to the cellular and colony scales. These levels of complexity are not independent instead they are tightly interrelated influencing each other directly. In this respect, stochastic P systems present an integrating multiscale modeling framework which explicitly specifies the molecular, cellular and colony levels in cellular systems in a relevant and understandable manner.

One of our research lines consists of the development of integrative modeling principles within the modeling framework of stochastic P systems. More specifically we will present some ideas on how to describe molecular species, cellular regions and compartments, molecular interactions, gene expression control and cell colonies. Our running example will be used to illustrate our modeling principles.

- **Molecular species:** These are specified as individual objects or strings of objects. Molecules with an internal structure that is relevant in the

functioning of the system are specified using strings. For example, gene operons with a linear structure consisting of promoters, operators, transcription/translation starting points, etc, otherwise molecular species are described using individual objects.

Table 1. Specification of the molecular species in the *lac operon*

Molecular Species	**Object**
RNA Polymerase	$RNAP$
Ribosome	Rib
Repressor	$LacI$
Activator	CRP CRP^*
LacZ product	$LacZ$
LacY product	$LacY$
LacA product	$LacA$
Lactose	$Lact$
Allolactose	$Allolac$
Glucose	$Gluc$
Glucose transport system	$Gluc$
Complex glucose transport system	$Gluc$-GTS
Complex lactose LacY product	$Lact$-$LacY$
Complex lactose LacZ product	$Lact$-$LacZ$
Complex lactose LacZ product	$Lact$-$LacZ$
Complex allolactose repressor	$Allolac$-$LacI$

Operon site	**Object**
Activator binding site	cap
Occupied activator binding site	cap^{CRP^*}
Repressor binding site	op
Occupied repressor binding site	op^{LacI}
lacZ gene	$lacZ$
lacY gene	$lacY$
lacA gene	$lacA$
lacZ mRNA	$mlacZ$
lacY mRNA	$mlacY$
lacA mRNA	$mlacA$

Running example: The different molecular species in our example will be specified according to this modeling principle. On the one hand, the proteins and complexes of proteins involved in the regulation and expression of the *lac operon* are specified as individual objects since we are not interested in their internal structure (Table 1). On the other hand, each component of the *lac operon* will be described using an object such that the *lac operon* structure is specified as a string containing these objects in the specific order they can be found in E. coli's genome (Figure 1).

- **Membranes:** Compartmentalization and membranes are fundamental in the structural organization and functioning of living cells. Membranes do not act as passive boundaries of cells and compartments; instead they play a key role in the regulation of the metabolism and information processing between the outside and the inside of compartments. P systems constitutes one of the few computational frameworks which explicitly specifies compartments and membranes. For instance, P systems have been used to study selective

uptake of molecules from the environment [20], signalling at the cell surface [12] and colonies of interacting bacteria which communicate by sending and receiving diffusing signals [2,21]. In general P system membranes are used to define relevant regions in cellular systems and therefore they do not always correspond to real cell membranes although normally they do.

Running example: In the *lac operon* gene regulation system there are two relevant regions. Namely, the bacterium surface where $LacY$ and GTS act as pumps transporting lactose and glucose into the cell and the aqueous interior or cytoplasm where the operon is located together with the different transcription factors and proteins. These two regions are represented using two membranes embedded one inside the other to described the structure of an E. coli bacterium (Figure 2).

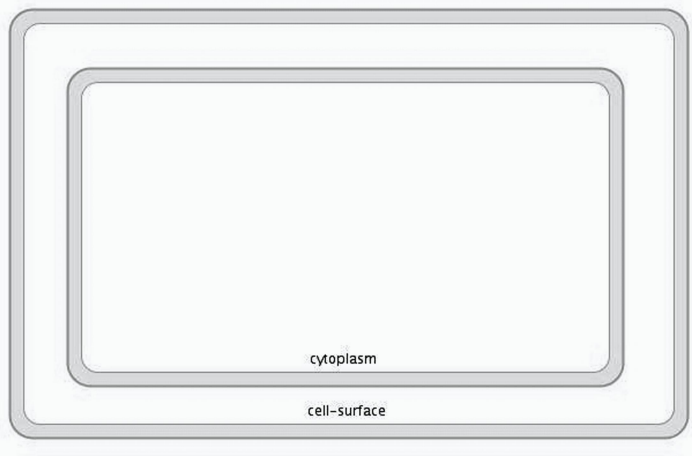


Fig. 2. Graphical representation of the membrane structure specifying an E. coli bacterium

- **Molecular processes consisting of protein-protein interactions and protein translocation:** Such processes are normally described in P systems using rewriting rules on multisets of objects. Our P system modeling framework aims at providing a comprehensive and relevant rule-based schema for the most common molecular interactions taking place in living cells. More specifically, our approach focuses on the transformation and degradation of molecular species, the formation and dissociation of complexes, and the basic processes of communication and transport between different compartments in cellular systems (Table 2).

  Running example: The protein-protein interactions in our gene regulation system are described using the rewriting rules on multisets of objects presented in Table 3. Rules r_{29}, r_{30}, r_{31} and r_{32} are examples of complex formation and dissociation rules. The degradation and dilution of different proteins

Table 2. P system rule-based schemas for the most common molecular interactions

Molecular Interaction	**P System Rules**
Transformation and Degradation	$[\ a\ ]_l \xrightarrow{c} [\ b\ ]_l \qquad [\ a\ ]_l \xrightarrow{c} [\ ]_l$
Complex formation and dissociation	$[\ a+b\ ]_l \xrightarrow{c_f} [\ c\ ]_l \quad [\ c\ ]_l \xrightarrow{c_d} [\ a+b\ ]_l$
Diffusion in and out	$a\ [\ ]_l \xrightarrow{c_{in}} [\ a\ ]_l \qquad [\ a\ ]_l \xrightarrow{c_{out}} a\ [\ ]_l$
Binding and debinding	$a\ [\ b\ ]_l \xrightarrow{c_{lb}} [\ c\ ]_l \qquad [\ c\ ]_l \xrightarrow{c_{ld}} a\ [\ b\ ]_l$
Recruitment and releasing	$a\ [\ b\ ]_l \xrightarrow{c_{rt}} c\ [\ ]_l \quad c\ [\ ]_l \xrightarrow{c_{rl}} a\ [\ b\ ]_l$

is specified in rules r_{22}, r_{23} and r_{24}. Finally, active uptake of glucose and lactose are modeled using the binding and releasing rules r_{27}, r_{28}, r_{33} and r_{34}.

- **Gene expression control:** The sensing of signals and the processing of the information they convey is performed in living cells through molecular interactions of the type presented in Table 2. The response of cells to these signals consists of the expression of appropriate proteins codified in specific genes. Gene expression control has been described in P systems using either rewriting rules on multisets of objects or rewriting rules on multisets of objects and strings according to the structural organization of the genes in the system under study. Tables 4 and 5 presents these two alternatives for the specification of the most important processes in gene expression control; transcription factor binding and debinding, transcription and translation.

  From a simplistic point of view the processes involved in transcription factor binding and debinding, transcription and translation can be represented by individual rewriting rules on multisets of objects (Table 4). Nevertheless, these processes are very complex and they consist of different stages like operator/promoter recognition by transcription factors and RNA polymerase, transcription/translation initiation/termination, elongation, etc. A more accurate and detailed description of all these processes is achieved by using rewriting rules on multisets of strings and objects of the form of the rules in Table 5.

  Running Example: The gene regulation control in the *lac operon* is modeled using the rewriting rules on multisets of objects and strings given in Table 3. More specifically, the binding and debinding of the activator and repressor to their corresponding binding sites is represented using rules r_3, r_4, r_7 and r_8. Transcription initiation in the presence and absence of the promoter site occupied by the activator CRP^* is specified using rules r_1, r_2, r_5 and r_6. The transcription of the structural genes *lacZ*, *lacY* and *lacA* is described by the rules r_9, r_{10}, r_{11} and r_{12}. Finally, translation and mRNA degradation is modeled with the rules r_{13} - r_{21}.
- **Cell colonies:** The last level of organization that has been represented using P systems consists of cellular systems where cells form colonies by interacting

Table 3. Lac Operon Regulation Rules

Nr.	Rule	Stochastic Constant
r_1 :	$[\ RNAP + \langle cap \rangle\ ]_b \xrightarrow{c_1} [\ \langle cap.RNAP \rangle\ ]_b$	$c_1 = 5 \times 10^{-3} min^{-1}$
r_2 :	$[\ \langle cap.RNAP \rangle\ ]_b \xrightarrow{c_2} [\ RNAP + \langle cap \rangle\ ]_b$	$c_2 = 1 min^{-1}$
r_3 :	$[\ CRP^* + \langle cap \rangle\ ]_b \xrightarrow{c_3} [\ \langle cap^{CRP^*} \rangle\ ]_b$	$c_3 = 16.6 min^{-1}$
r_4 :	$[\ \langle cap^{CRP^*} \rangle\ ]_b \xrightarrow{c_4} [\ CRP^* + \langle cap \rangle\ ]_b$	$c_4 = 10 min^{-1}$
r_5 :	$[\ RNAP + \langle cap^{CRP^*} \rangle\ ]_b \xrightarrow{c_5} [\ \langle cap^{CRP^*}.RNAP \rangle\ ]_b$	$c_5 = 0.2 min^{-1}$
r_6 :	$[\ \langle cap^{CRP^*}.RNAP \rangle\ ]_b \xrightarrow{c_6} [\ RNAP + \langle cap^{CRP^*} \rangle\ ]_b$	$c_6 = 1 min^{-1}$
r_7 :	$[\ LacI + \langle op \rangle\ ]_b \xrightarrow{c_7} [\ \langle op^{LacI} \rangle\ ]_b$	$c_7 = 166 min^{-1}$
r_8 :	$[\ \langle op^{LacI} \rangle\ ]_b \xrightarrow{c_8} [\ LacI + \langle op \rangle\ ]_b$	$c_8 = 0.1 min^{-1}$
r_9 :	$[\ \langle RNAP.op \rangle\ ]_b \xrightarrow{c_9} [\ \langle op.RNAP \rangle\ ]_b$	$c_9 = 3 min^{-1}$
r_{10} :	$[\ \langle RNAP.lacZ \rangle\ ]_b \xrightarrow{c_{10}} [\ \langle lacZ.RNAP \rangle ; \langle mlacZ \rangle\ ]_b$	$c_{10} = 0.78 min^{-1}$
r_{11} :	$[\ \langle RNAP.lacY \rangle\ ]_b \xrightarrow{c_{11}} [\ \langle lacY.RNAP \rangle ; \langle mlacY \rangle\ ]_b$	$c_{11} = 1.92 min^{-1}$
r_{12} :	$[\ \langle RNAP.lacA \rangle\ ]_b \xrightarrow{c_{12}} [\ RNAP + \langle lacA \rangle ; \langle mlacA \rangle\ ]_b$	$c_{12} = 4 min^{-1}$
r_{13} :	$[\ Rib + \langle mlacZ \rangle\ ]_b \xrightarrow{c_{13}} [\ \langle Rib.mlacZ \rangle\ ]_b$	$c_{13} = 0.12 min^{-1}$
r_{14} :	$[\ Rib + \langle mlacY \rangle\ ]_b \xrightarrow{c_{14}} [\ \langle Rib.mlacY \rangle\ ]_b$	$c_{14} = 0.12 min^{-1}$
r_{15} :	$[\ Rib + \langle mlacA \rangle\ ]_b \xrightarrow{c_{15}} [\ \langle Rib.mlacA \rangle\ ]_b$	$c_{15} = 0.12 min^{-1}$
r_{16} :	$[\ \langle Rib.mlacZ \rangle\ ]_b \xrightarrow{c_{16}} [\ Rib + LacZ + \langle mlacZ \rangle\ ]_b$	$c_{16} = 0.12 min^{-1}$
r_{17} :	$[\ \langle Rib.mlacY \rangle\ ]_b \xrightarrow{c_{17}} [\ Rib + LacY + \langle mlacY \rangle\ ]_b$	$c_{17} = 1.73 min^{-1}$
r_{18} :	$[\ \langle Rib.mlacA \rangle\ ]_b \xrightarrow{c_{18}} [\ Rib + LacA + \langle mlacA \rangle\ ]_b$	$c_{18} = 3.55 min^{-1}$
r_{19} :	$[\ \langle mlacZ \rangle\ ]_b \xrightarrow{c_{19}} [\ ]_b$	$c_{19} = 6 \times 10^{-3} min^{-1}$
r_{20} :	$[\ \langle mlacY \rangle\ ]_b \xrightarrow{c_{20}} [\ ]_b$	$c_{20} = 6 \times 10^{-3} min^{-1}$
r_{21} :	$[\ \langle mlacA \rangle\ ]_b \xrightarrow{c_{21}} [\ ]_b$	$c_{21} = 6 \times 10^{-3} min^{-1}$
r_{22} :	$[\ LacZ\ ]_b \xrightarrow{c_{22}} [\ ]_b$	$c_{22} = 6.9 \times 10^{-2} min^{-1}$
r_{23} :	$[\ LacY\ ]_b \xrightarrow{c_{23}} [\ ]_b$	$c_{23} = 6.9 \times 10^{-2} min^{-1}$
r_{24} :	$[\ LacA\ ]_b \xrightarrow{c_{24}} [\ ]_b$	$c_{24} = 6.9 \times 10^{-2} min^{-1}$
r_{25} :	$[\ LacY\ ]_b \xrightarrow{c_{25}} LacY\ [\ ]_b$	$c_{25} = 1 min^{-1}$
r_{26} :	$LacY\ [\ ]_b \xrightarrow{c_{26}} [\ LacY\ ]_b$	$c_{26} = 0.7 min^{-1}$
r_{27} :	$Lact\ [\ LacY\ ]_s \xrightarrow{c_{27}} [\ Lact\text{-}LacY\ ]_s$	$c_{27} = 10 min^{-1}$
r_{28} :	$Lact\text{-}LacY\ [\ ]_b \xrightarrow{c_{28}} LacY\ [\ Lact\ ]_b$	$c_{28} = 10 min^{-1}$
r_{29} :	$[\ Lact + LacZ\ ]_b \xrightarrow{c_{29}} [\ Lact\text{-}LacZ\ ]_b$	$c_{29} = 10 min^{-1}$
r_{30} :	$[\ Lact\text{-}LacZ\ ]_b \xrightarrow{c_{30}} [\ Allolac + LacZ\ ]_b$	$c_{30} = 10 min^{-1}$
r_{31} :	$[\ Allolac + LacI\ ]_b \xrightarrow{c_{31}} [\ Allolac\text{-}LacI\ ]_b$	$c_{31} = 1 min^{-1}$
r_{32} :	$[\ Allolac\text{-}LacI\ ]_b \xrightarrow{c_{32}} [\ Allolac + LacI\ ]_b$	$c_{32} = 10^{-4} min^{-1}$
r_{33} :	$Gluc\ [\ GTS\ ]_s \xrightarrow{c_{33}} [\ Gluc\text{-}GTS\ ]_s$	$c_{33} = 1 min^{-1}$
r_{34} :	$Gluc\text{-}GTS\ [\ ]_b \xrightarrow{c_{34}} GTS\ [\ Gluc\ ]_b$	$c_{34} = 10 min^{-1}$
r_{35} :	$GTS\ [\ CRP\ ]_b \xrightarrow{c_6} GTS\ [\ CRP^*\ ]_b$	$c_{35} = 6.9 \times 10^{-3} min^{-1}$
r_{36} :	$[\ CRP^*\ ]_b \xrightarrow{c_6} [\ ]_b$	$c_{36} = 0.069 min^{-1}$

and exhibiting coordinated behavior. The specification of the environment where the colony of cell is located cannot always be described by a single membrane since it is normally too big to be considered a well mixed volume or region where the Gillespie Algorithm can be applied. In this respect,

Table 4. P system rule-based schemas for gene expression control using multisets of objects

Molecular Interaction	**Rules on Multisets of Objects**
Transcription Factor Binding and Debinding	$[\, Tf + gene \,]_l \xrightarrow{c_{on}} [\, Tf\text{–}gene \,]_l$ $[\, Tf\text{–}gene \,]_l \xrightarrow{c_{off}} [\, Tf + gene \,]_l$
Transcription	$[\, gene \,]_l \xrightarrow{c_{tc}} [\, gene + rna \,]_l$
Translation	$[\, rna \,]_l \xrightarrow{c_{tl}} [\, rna + prot \,]_l$

Table 5. P system rule-based schemas for gene expression control using multisets of objects and strings

Molecular Interaction	**Rules on Multisets of Strings and Objects**
Transcription Factor Binding and Debinding	$[\, Tf + \langle op \rangle \,]_l \xrightarrow{c_{on}} [\, \langle op' \rangle \,]_l$ $[\, \langle op' \rangle \,]_l \xrightarrow{c_{off}} [\, Tf + \langle op \rangle \,]_l$
Transcription	$[\, \mathrm{RNAP} + \langle prom \rangle \,]_l \xrightarrow{c_{rb}} [\, \langle prom.\mathrm{RNAP} \rangle \,]_l$ $[\, \langle \overline{s}_0.w.\mathrm{RNAP}.s_N \rangle \,]_l \xrightarrow{c_{el}} [\, \langle s_N.\overline{s}_0.w.\overline{s}_N.\mathrm{RNAP} \rangle \,]_l$ $[\, \langle \overline{s}_0.w.\mathrm{RNAP}.s_t \rangle \,]_l \xrightarrow{c_{ter}} [\, \mathrm{RNAP} + \langle s_t \rangle;\ \langle \overline{s}_0.w.\overline{s}_t \rangle]_l$
Translation	$[\, \mathrm{Rib} + \langle \overline{s}_0 \rangle \,]_l \xrightarrow{c_{tli}} [\, \langle \overline{site}_0.\mathrm{Rib} \rangle \,]_l$ $[\, \langle \mathrm{Rib}.\overline{s}_N \rangle \,]_l \xrightarrow{c_{tle}} [\, \langle \overline{s}_N.\mathrm{Rib} \rangle \,]_l$ $[\, \langle \mathrm{Rib}.\overline{s}_t \rangle \,]_l \xrightarrow{c_{tlt}} [\, \mathrm{Rib} + \mathrm{Prot} + \langle \overline{s}_t \rangle \,]_l$

the environment is divided into a set of small regions that can be considered well mixed volumes. These regions are then connected according to a graph defining the topology of the environment. This structure has been termed multienvironment [11]. A cell colony is then specified as a collection of individual P systems representing individual cells distributed over the multienvironment. These P systems interact by passive or active transport rules using some of the specifications of molecular interaction described previously. The different regions in a multienvironment can also interact by passive diffusion rules of the following form:

$$[\, obj \,]_l - [\ ]_{l'} \xrightarrow{c} [\ ]_l - [\, obj \,]_{l'} \tag{3}$$

These rules are multiset rewriting rules that operate on two environments, one labeled l which is linked to another environment labeled l'. A multiset obj is removed from the first environment and placed in the second one. In this way, we are able to capture in a concise way the diffusion of signals from one region to another in a large environment. As well as objects, P systems

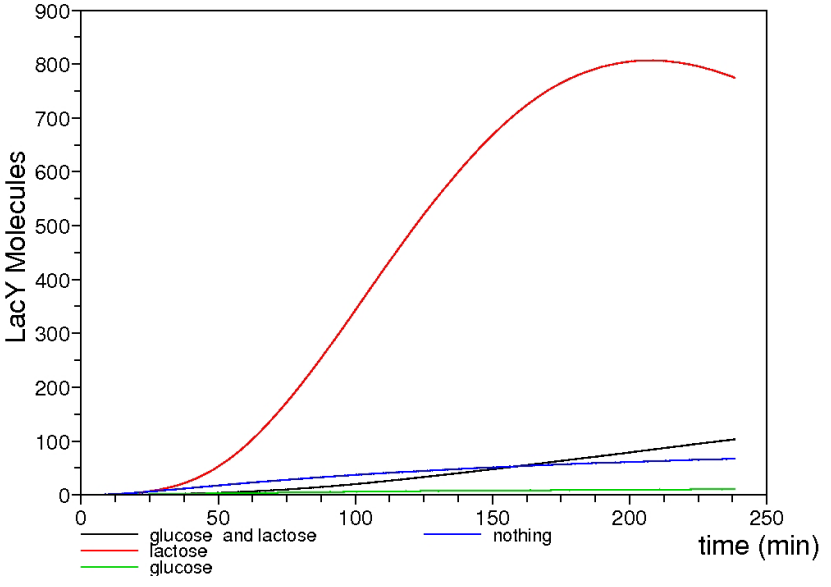


Fig. 3. Evolution over time of the average of the LacY protein products over a colony of 1000 bacteria for four different environmental conditions representing the presence and/or absence of glucose and/or lactose

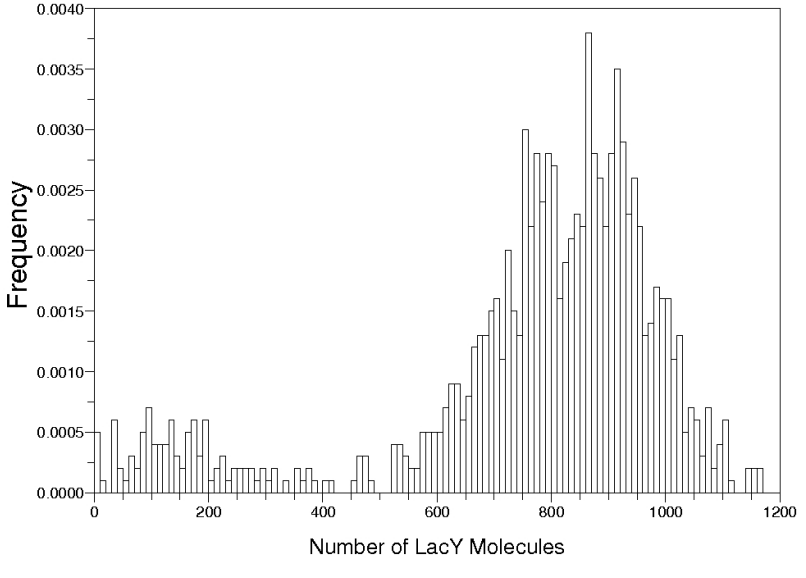


Fig. 4. Histogram representing the frequency of the number of LacY proteins over a colony of 1000 bacteria

representing individual cells can be moved from one environment to another using the following type of rules:

$$[\,[\,]_{l''}\,]_{l} - [\ ]_{l'} \xrightarrow{c} [\ ]_{l} - [\,[\,]_{l''}\,]_{l'} \tag{4}$$

When a rule of this type is applied, a membrane with label l'' and all its contents, objects and other membranes, are moved from an environment labeled l to another connected to it that must be labeled l'. This colony level specification of P systems was introduced in [2] and was used to model the quorum sensing system in the marine bacterium *Vibrio fischeri* from an artificial life perspective in [21].

Running Example: In our case study we have examined the behavior of a colony of 1000 bacteria located inside a single environment. Each bacterium was represented using the membrane structure in Figure 2, the rewriting rules in Table 3, the objects in Table 1 and the string in Figure 1. We have run simulations for four different environmental conditions representing the presence and/or absence of glucose and/or lactose.

We computed the average across the entire colony of the number of LacY protein products over time (Figure 3). Our results were in agreement with the known behavior of *E. coli* which prefers to consume glucose to lactose. Only in the absence of glucose and presence of lactose will *E. coli* fully express the *lac operon* and activate the processes involved in the uptake and metabolism of lactose.

In order to get a more detailed idea of the behavior of the colony we studied the frequency of the number of LacY protein products over all the bacteria in the case when only lactose is present in the environment (Figure 4). The number of LacY protein products over the colony is a bimodal distribution. Most bacteria fully express the *lac operon*, whereas a small but still noticeable fraction of the colony does not activate the uptake and metabolism of lactose. This type of behavior is characteristic of gene regulation systems with positive feedback loops [1].

5 Modularization

Cellular processes arise as emergent behavior from the interactions between many molecular species. It has been postulated that although these interactions are apparently messy they are arranged in a modular way. Our P system modeling framework supports the use of modules of rules to represent biological functions that are separable or orthogonal to some extend from the functioning of the rest of the system. A P system module is a set of rewriting rules of the forms previously introduced describing molecular processes occurring frequently in cell systems. A module is identified with a name and three sets of variables, V, representing the molecular species; C, the stochastic constants associated with each rules; and Lab, the labels of the compartments involved in the rules. Since a module is a set of rules starting from simple modules more complex modules

can be constructed by applying set union. Table 6 presents example P system modules describing the most common regulation mechanism in gene expression.

Following this idea one of our current research lines focuses on the automatic development of cellular models exhibiting a prefixed behavior by assembling P system modules automatically. Our methodology optimizes both the modular structure of the P systems and the stochastic constants associated with the rules. Specifically, our methodology consists of two nested genetic algorithms: the first one evolves the combination of modules or modular structure of the model whereas the second one optimizes the stochastic constants associated with the different rules in the modules. Our approach is incremental, by starting with simple predefined modules from an elementary library newly generated modules obtained by combining these elementary modules are added to the library after having been analyzed and validated. This allows us to develop more intricate and circuitous modular structures. Our methodology has been tested on three case studies, namely, molecular complexation, enzymatic reactions and autoregulation in transcriptional networks [22].

Table 6. Three examples of P system modules describing the most common regulation mechanisms in gene expression

Molecular process	**P System Module**
Constitutive Expression	$[\ gene\ ]_l \xrightarrow{c_1} [\ gene + rna\ ]_l$ $[\ rna\ ]_l \xrightarrow{c_2} [\ ]_l$ $[\ rna\ ]_l \xrightarrow{c_3} [\ rna + p\ ]_l$ $[\ p\ ]_l \xrightarrow{c_4} [\ ]_l$
Positive Regulation	$[\ a + gene\ ]_l \xrightarrow{c_1} [\ a.gene\ ]_l$ $[\ a.gene\ ]_l \xrightarrow{c_2} [\ a + gene\ ]_l$ $[\ a.gene\ ]_l \xrightarrow{c_3} [\ a.gene + rna\ ]_l$ $[\ rna\ ]_l \xrightarrow{c_4} [\ ]_l$ $[\ rna\ ]_l \xrightarrow{c_5} [\ rna + p\ ]_l$ $[\ p\ ]_l \xrightarrow{c_6} [\ ]_l$
Negative Regulation	$[\ gene\ ]_l \xrightarrow{c_1} [\ gene + rna\ ]_l$ $[\ r + gene\ ]_l \xrightarrow{c_2} [\ r.gene\ ]_l$ $[\ r.gene\ ]_l \xrightarrow{c_3} [\ r + gene\ ]_l$ $[\ rna\ ]_l \xrightarrow{c_4} [\ ]_l$ $[\ rna\ ]_l \xrightarrow{c_5} [\ rna + p\ ]_l$ $[\ p\ ]_l \xrightarrow{c_6} [\ ]_l$

In cellular systems, modularization does not only arise from chemical specificity, but is also determined by spatial localization of molecular species in different compartments. The P system modules introduced so far only describe modularity due to chemical specificity. No geometric information is associated with any components of P systems. Recently we have proposed to extend population P systems by using finite lattices on which individual P systems are distributed. These P systems communicate by sending and receiving objects according to rules of the following form:

$$[\ obj\ ]_l \overset{\mathbf{v}}{\Leftrightarrow} [\ \ ]_{l'} \xrightarrow{c} [\ ]_l \overset{\mathbf{v}}{\Leftrightarrow} [\ obj\ ]_{l'} \tag{5}$$

The application of a rule of the previous form in a P system Π_j located in position $\mathbf{p}$, moves the objects obj from the skin membrane l of Π_j to the skin membrane l' of P system $\Pi_{j'}$ located in position $\mathbf{p} + \mathbf{v}$. The stochastic constant c associated with the rule plays the same role as in the previous cases.

6 Conclusions and Future Work

In this paper we have presented P systems as a modeling approach within *executable biology* able to specify and simulate multiscale systems ranging from the molecular to the cell and colony level. The modeling principles used in P systems for the specification of molecular species, networks of interacting molecules, individual cells and collections of cells have been discussed. A running example consisting in an abstract gene regulation system based on the *lac operon* has been used to illustrate our approach. Our results show the characteristic bimodal behavior of a colony of bacterial cells with a positive feed back loop.

This framework is currently being extended and implemented in a software system with integrates stochastic simulation of multicellular P system models with analytic techniques based on model checking and automate model generation through the assembling of P systems modules. This framework is also being used to develop models of plant hormone transport in *Arabidopsis thaliana* and quorum sensing in *Pseudomonas aeruginosa*.

Acknowledgements. We would like to acknowledge EPSRC grant EP/E017215/1 and BBSRC grants BB/F01855X/1 and BB/D019613/1.

References

1. Alon, U.: Network motifs: theory and experimental approaches. Nature Reviews Genetics 8, 450–461 (2007)
2. Bernardini, F., Gheorghe, M., Krasnogor, N.: Quorum sensing P systems. Theoretical Computer Sci. 371, 20–33 (2007)
3. Besozzi, D., Cazzaniga, P., Pescini, D., Mauri, G., Colombo, S., Martegani, E.: Modeling and stochastic simulation of the Ras/cAMP/PKA pathway in the yeast Saccharomyces cerevisiae evidences a key regulatory function for intracellular guanine nucleotides pools. Journal of Biotechnology 133, 377–385 (2008)
4. Bianco, L., Fontana, F., Manca, V.: P systems with reaction maps. Intern. J. Foundations of Computer Sci. 17, 27–48 (2006)
5. Ciobanu, G., Pan, L., Păun, G., Pérez-Jiménez, M.J.: P systems with minimal parallelism. Theoretical Computer Sci. 378, 117–130 (2007)
6. Fisher, J., Henzinger, T.A.: Executable cell biology. Nature Biotechnology 25, 1239–1249 (2007)
7. Fontana, F., Manca, V.: Discrete solutions to differential equations by metabolic P systems. Theoretical Computer Sci. 372, 165–182 (2007)
8. Freund, R.: P systems working in the sequential mode on arrays and strings. Int. J. Found. Comput. Sci. 16, 663–682 (2005)
9. Gillespie, D.T.: Stochastic simulation of chemical kinetics. Annu. Rev. Phys. Chem. 58, 35–55 (2007)
10. Heiner, M., Gilbert, D., Donaldson, R.: Petri nets for systems and synthetic biology. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 215–264. Springer, Heidelberg (2008)
11. Krasnogor, N., Gheorghe, M., Terrazas, G., Diggle, S., Williams, P., Camara, M.: An appealing computational mechanism drawn from bacterial quorum sensing. Bulletin of the EATCS 85, 135–148 (2005)

12. Păun, A., Jesús Pérez-Jímenez, M., Romero-Campero, F.J.: Modeling signal transduction using P systems. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 100–122. Springer, Heidelberg (2006)
13. Păun, G.: Computing with membranes. J. Computer and System Sci. 61, 108–143 (2000)
14. Păun, G.: Membrane Computing: An Introduction. Springer, Heidelberg (2002)
15. Păun, G., Romero-Campero, F.J.: Membrane computing as a modeling framework. Cellular systems case studies. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 168–214. Springer, Heidelberg (2008)
16. Jesús Pérez-Jímenez, M., Romero-Campero, F.J.: P systems, a new computational modelling tool for systems biology. In: Priami, C., Plotkin, G. (eds.) Transactions on Computational Systems Biology VI. LNCS (LNBI), vol. 4220, pp. 176–197. Springer, Heidelberg (2006)
17. Pescini, D., Besozzi, D., Mauri, G., Zandron, C.: Dynamical probabilistic P systems. Intern. J. Foundations of Computer Sci. 17, 183–204 (2006)
18. Ptashne, M., Gann, A.: Genes and Signals. Cold Spring Harbor Laboratory Press (2002)
19. Regev, A., Shapiro, E.: The π-calculus as an abstraction for biomolecular systems. Modelling in Molecular Biology, 1–50 (2004)
20. Romero-Campero, F.J., Pérez-Jiménez, M.J.: Modelling gene expression control using P systems: the Lac Operon, a case study. BioSystems 91, 438–457 (2008)
21. Romero-Campero, F.J., Pérez-Jiménez, M.J.: A model of the quorum sensing system in Vibrio fischeri using P systems. Artificial Life 14, 1–15 (2008)
22. Romero-Campero, F.J., Cao, H., Cámara, M., Krasnogor, N.: Structure and parameter estimation for cell systems biology models. In: Proc. of the Genetic and Evolutionary Computation Conference, Atlanta, USA, pp. 331–338 (2008)
23. Romero-Campero, F.J., Twycross, J., Cámara, M., Bennett, M., Gheorghe, M., Krasnogor, N.: Modular assembly of cell systems biology models using P systems (submitted)

On the Qualitative Analysis of Conformon P Systems

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

[1] University of Uppsala
parosh@it.uu.se
[2] Università di Genova, Italy
giorgio@disi.unige.it
[3] Université Libre de Bruxelles, Belgium
lvbegin@ulb.ac.be

Abstract. We study computational properties of conformon P systems, an extension of P systems in which symbol objects are labeled 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 labeled 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.

⋆ Research fellow supported by the Belgian National Science Foundation.

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 78–94, 2009.

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 $\frac{e}{m}A$, 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 $A \xrightarrow[m]{e} B$, 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 $m \xhookrightarrow{p} n$ 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*Psystem 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 $r = A \xrightarrow[m]{e} B$ 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 $r = m \xhookrightarrow{p} n$ 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 $r = A \xrightarrow[m]{e} B$ 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 $r = m \xhookrightarrow{p} n$ 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 $r = \xrightarrow[m]{e} A$ 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

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

and the set of predecessor configurations is defined as

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

Notice that the transition relation $\Rightarrow$ defines an interleaving semantics for a cPsystem 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 cPsystem with two membranes m_1 and m_2 and $N = \{A, B, C\}$, and with the rules $\xrightarrow[m_1]{1} A$, $A \xrightarrow[m_1]{1} B$, 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 cPsystems

In [5] Frisco introduced the class of cPsystems with *bounded values* in which the only type of admitted creation rules have the form $\xrightarrow[m]{0} A$, 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 cPsystem 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 cPsystems 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 cPsystems.

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 cPsystems 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 cPsystems. Let us first define the reachability problem.

Definition 2 (Reachability problem)
The reachability problem is defined as follows: Given a c*Psystem* $\Pi = (V, N, R, \mu_0)$ *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*Psystems)
The reachability problem (w.r.t. relation $\Rightarrow$*) is decidable for any* c*Psystem.*

Proof. The proof is based on a reduction of reachability of configuration μ_1 in a cPsystem 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 $r = A \xrightarrow[m]{2} B$. 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 cPsystem 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*Psystem $\Pi = (V,N,R,\mu_0)$ 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*Psystems.*

Proof. The encoding of a counter machine M in cPsystems 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 cPsystem 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 cPsystems. □

3.1 A Syntactic Fragments of *c*Psystems

In this section we show that checking safety properties can be decided for a fragment of cPsystems with a restricted form of passage rules in which conditions are only defined by lower bound constraints.

Definition 5 (Restricted *c*Psystems). *We call* restricted *the fragment of* c*Psystems 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 cPsystems with an arbitrary number of conformons. To prove the result we adopt the methodology proposed in [1], i.e., we first show that restricted cPsystems 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*Psystems 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 $m \overset{p}{\hookrightarrow} n$ with $p(y) \overset{\text{def}}{=} y \geq e$ 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*Psystems and any upward closed set (w.r.t.* $\sqsubseteq$*) S of configurations, the set of predecessor configurations of S, namely* $Pre(S) \overset{\text{def}}{=} \{\mu \mid \mu \Rightarrow \mu',\ \mu' \in S\}$*, is upward closed.*

It is important to notice that the last two properties do not hold for full cPsystems. 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*Psystem 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*Psystem $\Pi = (V, N, R, \mu_0)$, 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 inclusion 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*Psystems) *The coverability problem is decidable for restricted* c*Psystems.*

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*Psystems with other models used in the concurrency field, namely the nested Petri nets of [8] and the constrained multiset rewriting systems (CMRS) of [2].

4.1 *c*PSystems 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 emptiness 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*Psystems as nested Petri nets.* In this section we show that *c*Psystems 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*Psystem $\Pi = (V, N, R, \mu_0)$. 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 $CREAT E_r$ 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 $\{\!\{CREAT E_r\}\!\}$ and post-set $\{\!\{CREAT E_r, CONF\}\!\}$. We assume here that $CREAT E_r$ 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 $r = m \overset{p}{\hookrightarrow} n$ 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 occurrence 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 $x = 0, \ldots, x = e$.

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 cPsystem Π with $V = \{A, B\}$, $N = \{M, L, P\}$, the creation rule $r = \frac{4}{M} A$, the internal rule $INT = A \frac{3}{M} B$, 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 cPsystems Π is shown in Fig. 1. We use here circles to denote places, rectangles to denote transitions, an arrow from a circle to a rectangle to denote places

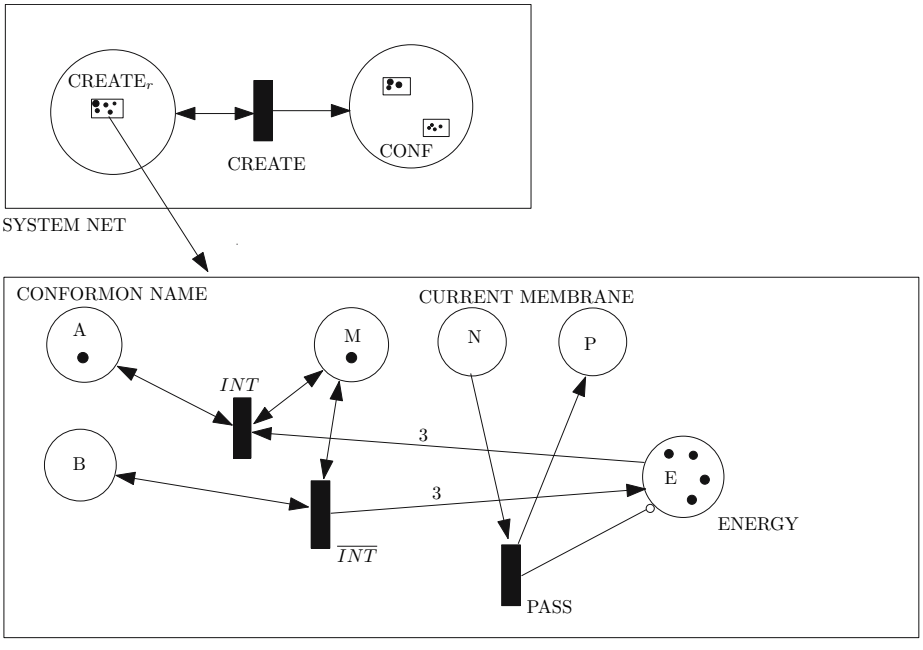


Fig. 1. Example of nested Petri net

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 represented 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 modeled 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 modeled 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 cPsystems [5]. From the previous observations,

restricted *c*Psystems 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*Psystems.

The connection between *c*Psystems 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*Psystems 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).

4.2 Restricted *c*PSystems vs. CMRS

Restricted *c*Psystems can also be modeled 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*Psystems as CMRS.* A cP-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 cP-configuration μ is obtained by assigning a distinct identifier 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 identifiers (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 cPsystem 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:

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

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:

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

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 cPsystems. The connection between cPsystems 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.

References

1. Abdulla, P.A., Čerāns, K., Jonsson, B., Yih-Kuen, T.: General decidability theorems for infinite-state systems. In: Proc. LICS 1996, pp. 313–321 (1996)
2. Abdulla, P.A., Delzanno, G.: On the coverability problem for constrained multiset rewriting systems. In: AVIS 2006 (ETAPS-workshop) (2006)
3. Dang, Z., Ibarra, O.H., Li, C., Xie, G.: On model-checking of P systems. In: Calude, C.S., Dinneen, M.J., Păun, G., Jesús Pérez-Jímenez, M., Rozenberg, G. (eds.) UC 2005. LNCS, vol. 3699, pp. 82–93. Springer, Heidelberg (2005)
4. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theoretical Computer Sci. 256, 63–92 (2001)
5. Frisco, P.: The conformon-P system: a molecular and cell biology-inspired computability model. Theoretical Computer Sci. 312, 295–319 (2004)
6. Frisco, P., Corne, D.W.: Dynamics of HIV infection studied with cellular automata and conformon-P systems. BioSystems 91, 531–544 (2008)
7. Higman, G.: Ordering by divisibility in abstract algebras. London Math. Soc. 3, 326–336 (1952)
8. Lomazova, I.A.: Nested Petri nets. Fundamenta Informaticae 43, 195–214 (2000)
9. Lomazova, I.A., Schnoebelen, P.: Some decidability results for nested Petri nets. In: Ershov Mem. Conf., pp. 208–220 (1999)
10. Păun, G.: Computing with membranes. J. Computer and System Sci. 61, 108–143 (2000)
11. Reisig, W.: Petri Nets: An Introduction. Springer, Heidelberg (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, Blvd. Carol I no.11, 700506 Iaşi, Romania
gabriel@info.uaic.ro

Abstract. 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

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 95–107, 2009.

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 labeled 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 $\Pi = (O, \mu, w_1, \dots w_m, R_1, \dots, R_m, i_o)$ where:

- O is an alphabet of objects;
- μ is a membrane structure, with the membranes labeled by natural numbers $1, \dots, m$, in a one-to-one manner;
- w_i are multisets over O associated with the regions $1, \dots, m$ defined by μ;
- $R_1, \dots, R_m$ are finite sets of rules associated with the membranes with labels $1, \dots, m$; the rules have the form $u \to v$, 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 $R_1^{\Pi}, \dots, R_m^{\Pi}$ for the sets of rules $R_1, \dots, R_m$ of the P system Π.

We consider a multiset w over a set S to be a function $w : S \to \mathbf{N}$. When describing a multiset characterized 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 $w + w' : S \to \mathbf{N}, (w + w')(s) = w(s) + w'(s)$. 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

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

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

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 labeled by i (if it exists) and $children(i)$ for the set of labels indicating the children of the membrane labeled 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 $\mathcal{R} : R_i^{\Pi} \to \mathbf{N}$ 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 $\mathcal{R} : R_i^{\Pi} \to \mathbf{N}$ 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, labeled 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* $\mathcal{R} : R_i^{\Pi} \to \mathbf{N}$ *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}, $i \in \{1, \ldots, m\}$.*

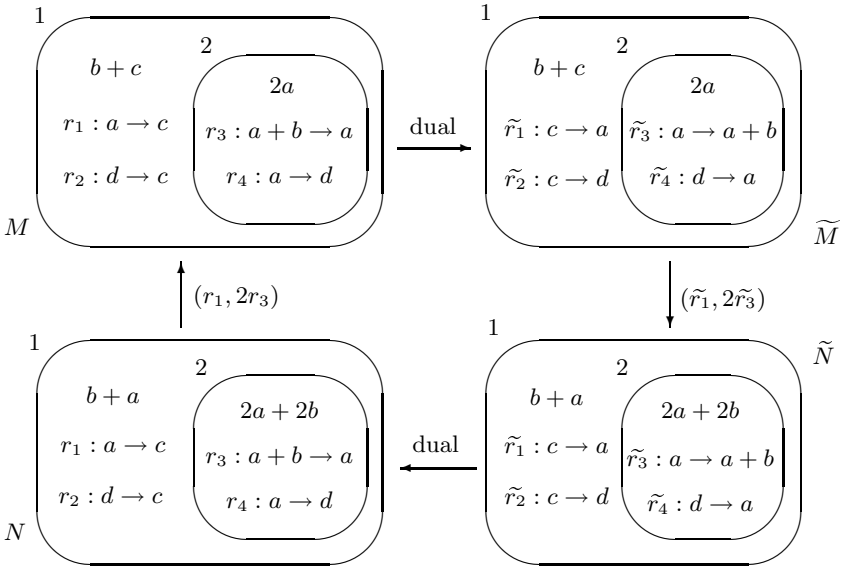
A system of multisets of rules $\mathcal{R}$ is called *valid, maximally valid* or *reversely valid* in the configuration M if each $\mathcal{R}_i$ is valid, maximally valid or reversely valid in the multiset $w_i(M)$, which, we recall, is the multiset contained in the inner membrane of configuration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

If $\widetilde{M} \stackrel{\widetilde{\mathcal{R}}}{\rightarrow}_{\widetilde{mpr}} \widetilde{N}$ 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. $\widetilde{r} = u \rightarrow v \in R_i^{\widetilde{\Pi}}$ if and only if $r : v \rightarrow u \in R_i^{\Pi}$;
2. $\widetilde{r} : u \rightarrow (v, out) \in R_i^{\widetilde{\Pi}}$ if and only if $r : v \rightarrow (u, in_i) \in R_{parent(i)}^{\Pi}$;
3. $\widetilde{r} : u \rightarrow (v, in_j) \in R_i^{\widetilde{\Pi}}$ if and only if $r : v \rightarrow (u, out) \in R_j^{\Pi}$, $i = parent(j)$;

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:*

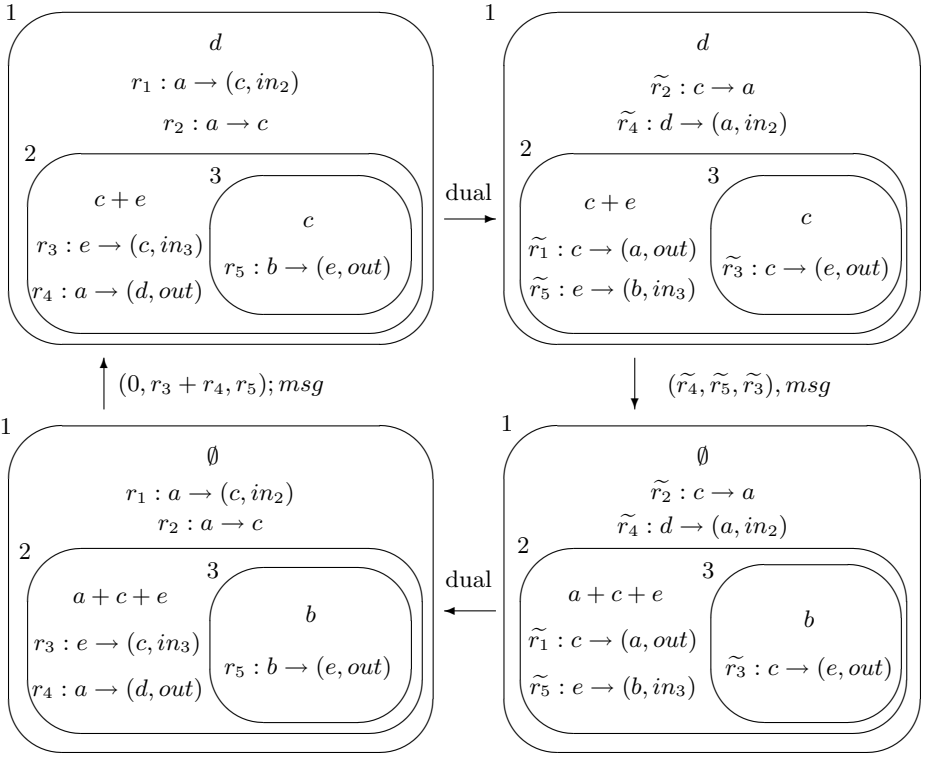
$$\widetilde{\widetilde{\Pi}} = \Pi$$

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

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* $r : u \to (v, in_i) \in R^{\Pi}_{parent(i)}$ *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* $r : u \to (v, out) \in R^{\Pi}_j$ *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 $0_{R^{\widetilde{\Pi}}_1}$ or $\widetilde{r_4}$; $\widetilde{\mathcal{R}_2}$ equal to $0_{R^{\widetilde{\Pi}}_2}$, $\widetilde{r_1}$, $\widetilde{r_5}$ or $\widetilde{r_1} + \widetilde{r_5}$; $\widetilde{\mathcal{R}_3}$ equal to $0_{R^{\widetilde{\Pi}}_3}$ 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 $\widetilde{\mathcal{R}_2} = 0_{R^{\widetilde{\Pi}}_2}$ 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 $c \to (e, out) \in R^{\widetilde{\Pi}}_3$. 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 $\widetilde{\mathcal{R}_2} = 0_{R^{\widetilde{\Pi}}_2}$.

In P systems with communication rules we work with both rewriting and message sending. We have presented two semantics for rewriting in Section 3: $\to_{mpr}$ (maximally parallel rewriting) and $\to_{\widetilde{mpr}}$ (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 $w : \Omega \to \mathbf{N}$ 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*,* $M \to_{msg} N$ *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*

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

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

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

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:

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

$$rhs^{obj}(\mathcal{R}_i)(a) = \sum_{r:u \rightarrow v \in R_i^{\Pi}} R_i(r) \cdot v(a),$$

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

$$rhs^{out}(\mathcal{R}_i)(a) = \sum_{r:u \rightarrow (v,out) \in R_i^{\Pi}} R_i(r) \cdot v(a),$$

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

$$rhs^{in_j}(\mathcal{R}_i)(a) = \sum_{r:u \rightarrow (v,in_j) \in R_i^{\Pi}} R_i(r) \cdot v(a).$$

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 $lhs^{in_j}(\mathcal{R}_i) = rhs^{out}(\widetilde{\mathcal{R}_j})$, $rhs^{in_j}(\mathcal{R}_i) = lhs^{out}(\widetilde{\mathcal{R}_j})$;
- $lhs(\mathcal{R}_i) = lhs^{obj}(\mathcal{R}_i) + lhs^{out}(\mathcal{R}_i) + \sum_{j \in children(i)} lhs^{in_j}(\mathcal{R}_i)$.

Now we can prove the statements of this Proposition. We prove only the first one; the proof of the second one is similar. If $M \xrightarrow{\mathcal{R}}_{mpr} \rightarrow_{msg} N$ then there exists an intermediate configuration P such that $M \xrightarrow{\mathcal{R}}_{mpr} P$ and $P \rightarrow_{msg} N$. 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 $P \rightarrow_{msg} N$ 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 $v \leq w_i(\widetilde{N}) - lhs(\widetilde{\mathcal{R}_i})$ it means that $lhs(r) \leq w_i(M) - lhs(\mathcal{R}_i)$, 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 $v \leq w_i(\widetilde{N}) - lhs(\widetilde{\mathcal{R}_i})$ then again $lhs(r) \leq w_i(M) - lhs(\mathcal{R}_i)$ (contradiction). The third situation leads to the same contradiction. Thus, there exists an intermediate configuration Q in $\widetilde{\Pi}$ such that $\widetilde{N} \overset{\widetilde{\mathcal{R}}}{\to}_{\widetilde{mpr}} Q$. We have to show that $Q \to_{msg} \widetilde{M}$, 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

$$lhs(\mathcal{R}_i) = lhs^{obj}(\mathcal{R}_i) + lhs^{out}(\mathcal{R}_i) + \sum_{j \in children(i)} lhs^{in_j}(\mathcal{R}_i). \qquad \square$$

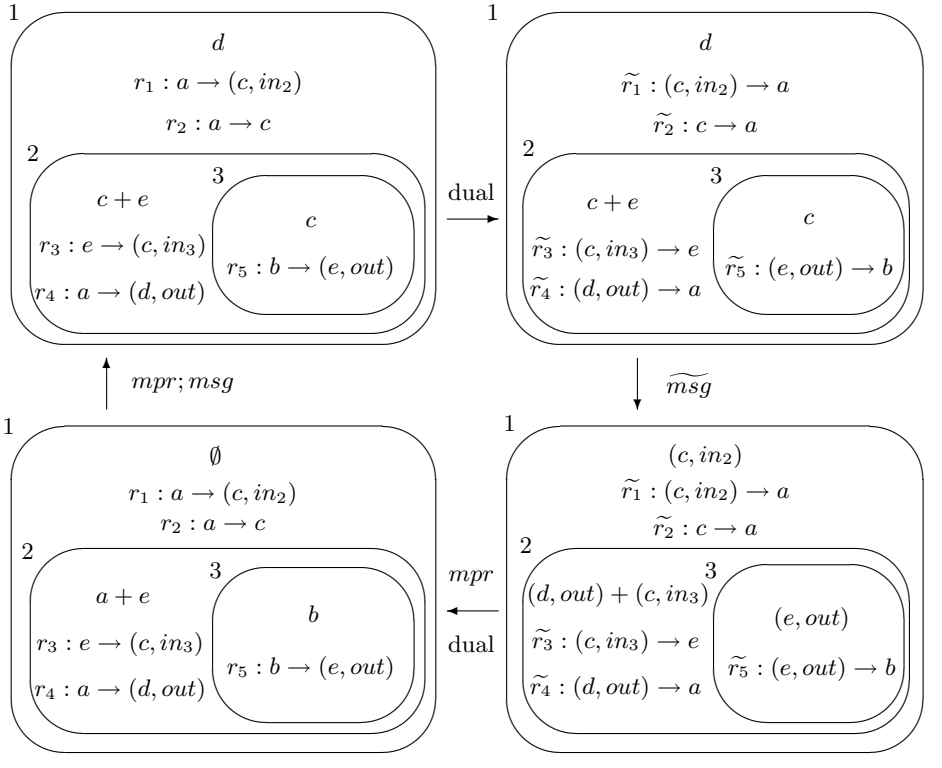
5 An Alternative Approach

Another way to reverse a computation $N \overset{\mathcal{R}}{\to}_{mpr} \to_{msg} M$ 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 $a \to (b, out)$ yields $(b, out) \to a$, 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

$$M \overset{\mathcal{R}}{\to}_{mpr} N \to_{msg} P \text{ if and only if } \widetilde{P} \to_{\widetilde{msg}} \widetilde{N} \overset{\widetilde{\mathcal{R}}}{\to}_{\widetilde{mpr}} \widetilde{M}$$

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



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

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

References

1. Barretta, C., Hunt III, H.B., Marathea, M.V., Ravic, S.S., Rosenkrantz, D.J., Stearns, R.E., Thakurd, M.: Predecessor existence problems for finite discrete dynamical systems. Theoretical Computer Sci. 386, 3–37 (2007)
2. Leporati, A., Zandron, C., Mauri, G.: Reversible P systems to simulate Fredkin circuits. Fundamenta Informaticae 74, 529–548 (2006)
3. Morita, K.: Reversible computing and cellular automata – A survey. Theoretical Computer Sci. 395, 101–131 (2008)
4. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)

Solving PP-Complete and #P-Complete Problems by P Systems with Active Membranes

Artiom Alhazov[1], Liudmila Burtseva[1], Svetlana Cojocaru[1], and 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

Abstract. 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 a 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. The implication of this result to the **PP** complexity class is discussed and compared to known results about $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP}$.

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 an 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) 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 in [4], using division for non–elementary membranes and three electrical charges. This result was improved 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

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 108–117, 2009.

division rules for elementary membranes (see, e.g., [8], [6], [3], [7], [9] and their references).

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 [11] for a presentation of Complexity Theory of counting problems.

Let us cite [12] 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 [13] 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.

In Section 4 we recall the definition of **PP** (the probabilistic polynomial time complexity class) and present an approach to solving the problems in **PP**.

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 labeled 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 labeled 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., one where 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 a 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 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., [8]. The definition below generalizes it from decisional problems to 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* to 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., [8], generalized from decisional problems to 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).

In this paper we do not need division, dissolution or rules that bring an object inside a membrane, but 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 [15], was first defined in [10] 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 $\sigma : \{1, \cdots, n\} \to \{1, \cdots, n\}$. The permanent of a matrix $A = (a_{i,j})_{1 \le i,j \le n}$ 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 Main Result

Theorem 1. *The problem of computing the 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 $\Sigma(n) = \{\langle i, j \rangle \mid 1 \le i \le n,\ 1 \le j \le n\}$, 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 $i_{\Pi(n)} = 2$.

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

and the rules are presented and explained below.

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

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

Division of the elementary membrane for Nn times.

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

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

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

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

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

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

Erase the chosen column, except the chosen element.

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

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, the 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 technical reasons: to reset to 0 the polarization of membranes that had polarization 1 after the first Nn steps).

It should be noted that the 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 a polynomial number of steps without having to recall from [1] rules sending objects out that work in parallel.

4 Attacking PP Complexity Class

The probabilistic polynomial complexity class **PP**, also called Majority-P, has been introduced in [2]. It is the class of **decision** problems solvable by a probabilistic Turing machine in polynomial time, with an error probability of less than 1/2 for all instances, see also [14]. It is known that $\mathbf{PP} \supseteq \mathbf{NP} \cup \mathbf{co} - \mathbf{NP}$, and the inclusion is strict if $\mathbf{P} \neq \mathbf{NP}$. Therefore, showing a solution to a **PP**-complete problem by P systems with active membranes without division of non-elementary membranes and without membrane creation would improve the best known results relating P systems to $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP}$.

In this section we show a way to do this, paying a small price of post-processing. We recall that the framework of solving decisional problems by P systems with active membranes includes two encoding functions (computing the description of a P system from the size of the problem instance and computing the input multiset from the instance of the problem). Unlike a more general case of solving computational problems, there was no need for the decoding function, since the meaning of objects `yes` and `no` sent to the environment was linked with the answer. While the decoding function was necessary for extending the framework for the computational problems (computing the answer to the instance of the problem from the output multiset of a P system in polynomial time), we would like to underline that it is useful even for the decisional problems.

It is not difficult to see that the problem "given a matrix A of size n, is $Perm(A) > n!/2$?" is **PP**-complete. Hence, we only have to compare the result of the computation of the matrix permanent with $n!/2$. Doing it by usual P systems with active membranes would need a non-polynomial number of steps. We can propose two approaches.

- Generalizing rules of type (a) to cooperative ones. It would then suffice to generate $n!/2$ copies of a new object z, then erase pairs of o and z and finally check if some object o remains. However, this class of P systems is not studied.
- Consider, as before, the number of objects o as the result of the computation of a P system. Use the decoding function

$$dec(x) = \begin{cases} \texttt{no} & , x \leq n!/2, \\ \texttt{yes} & , x > n!/2. \end{cases}$$

  The function dec can obviously be computed in polynomial time.

5 Discussion

In this paper we presented a solution to the problem of computing the 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 $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP}$. Indeed, paying the small price of using the decoding function also for decisional problem this approach allows to solve problems in the class **PP**, which is strictly larger than that (assuming $\mathbf{P} \neq \mathbf{NP}$).

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.

References

1. Alhazov, A., Pan, L., Păun, G.: Trading polarizations for labels in P systems with active membranes. Acta Informaticae 41, 111–144 (2004)
2. Gill, J.: Computational complexity of probabilistic Turing machines. SIAM Journal on Computing 6, 675–695 (1977)
3. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A fast P system for finding a balanced 2-partition. Soft Computing 9, 673–678 (2005)
4. Păun, G.: P systems with active membranes: Attacking NP–complete problems. J. Automata, Languages and Combinatorics 6, 75–90 (2001)
5. Păun, G., Suzuki, Y., Tanaka, H., Yokomori, T.: On the power of membrane division in P systems. Theoretical Computer Sci. 324, 61–85 (2004)
6. Pérez-Jiménez, M.J., Riscos-Núñez, A.: Solving the subset-sum problem by active membranes. New Generation Computing 23, 367–384 (2005)
7. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: Complexity classes in cellular computing with membranes. Natural Computing 2, 265–285 (2003)
8. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: Computationally hard problems addressed through P systems. In: Ciobanu, G., et al. (eds.) Applications of Membrane Computing, pp. 315–346. Springer, Heidelberg (2006)
9. Pérez Jiménez, M.J., Romero Campero, F.J.: Attacking the common algorithmic problem by recognizer P systems. In: Margenstern, M. (ed.) MCU 2004. LNCS, vol. 3354, pp. 304–315. Springer, Heidelberg (2005)
10. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Sci. 8, 189–201 (1979)
11. Wegener, I.: Complexity Theory: Exploring the Limits of Efficient Algorithms. Springer, Heidelberg (2005)
12. Williams, R.M., Wood, D.H.: Exascale computer algebra problems interconnect with molecular reactions and complexity theory. DIMACS Series in Discrete Mathematics and Theoretical Computer Sci. 44, 267–275 (1999)
13. `http://en.wikipedia.org/wiki/Permanent` (updated 05.05.2008)
14. `http://en.wikipedia.org/wiki/PP_complexity` (updated 09.09.2008)
15. `http://en.wikipedia.org/wiki/Sharp-P` (updated 13.12.2007)

Fast Synchronization in P Systems

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

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
str. Academiei 5, MD-2028, Chişinău, Moldova
artiom@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] LACL, Département Informatique, Université Paris Est
61 av. Général de Gaulle, 94010 Créteil, France
verlan@univ-paris12.fr

Abstract. We consider the problem of synchronizing the activity of all membranes of a P system. After pointing 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 for P systems. One algorithm is non-deterministic and works in $2h+3$ steps, the other is deterministic and works in $3h+3$ steps, 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 who stands in one end of the line, see [2,5,4,9,10,11]. 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; 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

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 118–128, 2009.

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, which 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 recognized 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 $3h + 3$ steps 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

$$\Pi = (O, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n), \text{ where:}$$

1. O is a finite alphabet of symbols called objects,
2. μ is a membrane structure consisting of n membranes labeled 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 $u \to v_1, tar_1; v_2, tar_2; \ldots; v_m, tar_m$, where $u \in O^+$, $v_i \in O$, and $tar_i \in \{in, out, here, in!\}$.

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 $u \to v_1, tar_1; v_2, tar_2; \ldots; v_m, tar_m$ 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. With a successful computation a result can be associated, but in what follows we are interested in the computation itself, not in any result of it.

A *transitional P system with promoters and inhibitors* is a system as defined as in the previous definition, where the set of rules may contain rules of the form

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

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 that 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.

We formulate 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 $i \in \{2, \ldots, n\}$, 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 The 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 the 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 a symbol 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 the corresponding node.

Now let us present the system in details.

Let $\Pi = (O, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n)$ be the P system to be synchronized. 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 $w_1 = \{S_1\}$ and that all other membranes but the skin one 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:

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

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

- *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 traveling up until it reaches the root node (d steps).

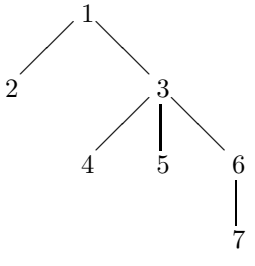
- *All nodes inner nodes will be marked by $\bar{S}$ and the leaves will be marked by S.* Indeed, rule (2) permits to implement this behavior.
- Let $d+2$, $0 \leq d \leq h$ be the moment when the root stops the guess of the tree height (the second rule from (3) has been applied). At this moment the contents of w_1 is $S_3 b^d$ and c starts to be propagated. Now consider any node x except the root. Then:

  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 subtraction.

  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. 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 nondeterministic, 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 The 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.

The P system $\Pi = (O, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n)$ 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:

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

Decrement:

$$S'' b \to S'' \qquad S''' a \to S''' \tag{25}$$
$$S'' \to F \mid_{\neg b} \qquad S''' \to F \mid_{\neg a} \tag{26}$$

Root decrement:

$$S_4 b \to S_4 \qquad S_4 \to F \mid_{\neg b} \tag{27}$$

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 the same membrane structure as the system from Example 1. The evolution of the system is as follows:

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 interesting property that after $2h$ steps either the system synchronizes and the object F is introduced, or an object # will be present in some membrane.

The results obtained in this article rely on a rather strong target indication, *in*!, which sends an object to all inner membranes. Such a synchronization was already considered in neural-like P systems where it corresponds to the target *go*. It would be interesting to investigate what happens if this target is not used. 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 this 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.

References

1. Bernardini, F., Gheorghe, M., Margenstern, M., Verlan, S.: How to synchronize the activity of all components of a P system? In: Vaszil, G. (ed.) Proc. Intern. Workshop Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, Hungary, pp. 11–22 (2007)
2. Goto, E.: A minimum time solution of the firing squad problem. Harward Univ. Course Notes for Applied Mathematics, 298 (1962)
3. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. American Mathematical Society 7, 48–50 (1956)
4. Mazoyer, J.: A six-state minimal time solution to the firing squad synchronization problem. Theoretical Science 50, 183–238 (1987)
5. Minsky, M.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
6. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
7. Prim, R.C.: Shortest connection networks and some generalizations. Bell System Technical Journal 36, 1389–1401 (1957)
8. Spellman, P.T., Sherlock, G.: Reply whole-cell synchronization - effective tools for cell cycle studies. Trends in Biotechnology 22, 270–273 (2004)
9. Umeo, H., Maeda, M., Fujiwara, N.: An efficient mapping scheme for embedding any one-dimensional firing squad synchronization algorithm onto two-dimensional arrays. In: Bandini, S., Chopard, B., Tomassini, M. (eds.) ACRI 2002. LNCS, vol. 2493, pp. 69–81. Springer, Heidelberg (2002)
10. Schmid, H., Worsch, T.: The firing squad synchronization problem with many generals for one-dimensional CA. In: Proc. IFIP TCS 2004, pp. 111–124 (2004)
11. Yunès, J.-B.: Seven-state solution to the firing squad synchronization problem. Theoretical Computer Sci. 127, 313–332 (1994)
12. The P systems web page, `http://ppage.psystems.eu`

Membrane Systems Using Noncooperative Rules with Unconditional Halting

Markus Beyreder and Rudolf Freund

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

Abstract. We consider membrane systems using noncooperative rules, but considering computations without using halting conditions. As results of a computation we take the contents of a specified output membrane/cell in each transition 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 membrane systems using noncooperative rules turns out to be equivalent to that of Lindenmayer 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 transition step, no matter whether this computation will ever halt or not, eventually taking only results completely consisting of terminal objects. In every transition step, we apply the traditional maximal parallelism. Other transition modes could be considered, too, but, for example, applying the sequential 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.

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 129–136, 2009.

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) L systems. Section 3 is dedicated to the definition of tissue P systems with noncooperative rules working in the maximally parallel 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) L 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 L 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 transition 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 L 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 L 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*

$$\Pi = (n, V, T, R, C_0, i_0)\,, \text{ where}$$

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*

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

   for $i \in [1..n]$, $a \in V$ *as well as* $b_j \in V$ *and* $h_j \in [1..n]$, $j \in [1..k]$, $k \geq 0$*;*
5. $C_0 = (w_1, \ldots, w_n)$*, 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 $(a, i) \to (b_1, h_1) \ldots (b_k, h_k)$ 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 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 $Ps_u(\Pi)$. 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 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 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 O 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 $\{a^{2^n} \mid n \geq 0\}$ 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 $Ps_u(\Pi) = Ps(L(G))$ and $N(G) = N_u(\Pi)$.

We 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); with $N(\Pi)$ denoting

the set of non-negative integers generated by Π with halting computations we obtain the following:

- if $w = \lambda$, then $N(\Pi) = \{0\}$;
- if R is empty, then $N(\Pi) = \{|w|\}$;
- if $w \neq \lambda$ and R contains the rule $a \to \lambda$, then $N(\Pi) = \{0\}$, 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., $N(\Pi)$ 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* $n \geq 1$, $PsE0L = PsEOtP_n(ncoo, max, u) = PsEOtP_*(ncoo, max, u)$.

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

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 $a \in V \times [1..n]$).

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 computations in Π and derivations in G we immediately obtain $Ps(h(L(G))) = Ps_u(\Pi)$.

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

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

Corollary 1. *For all* $n \geq 1$, $NE0L = NEOtP_n(ncoo, max, u) = NEOtP_*(ncoo, max, u)$.

Proof. Given an E0L system G, we construct the corresponding extended tissue P system Π as above in Theorem 1; then we immediately infer $N(G) = N_u(\Pi)$. On the other hand, given an extended tissue P system Π, by the constructions elaborated in Theorem 1, we obtain $N_u(\Pi) = N(G') = \{|x| \mid x \in h(L(G))\}$ and therefore $N_u(\Pi) \in NE0L$. □

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

Proof. This result immediately follows from the constructions elaborated in Theorem 1 with the specific restriction that for proving the inclusion $PsOtP_1(ncoo, max, u) \subseteq Ps0L$ 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 $Ps_u(\Pi) = L(G) \in Ps0L$. 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 (in the case $x = \lambda$ the rule a/x usually is called a symport rule, yet we do not take into account this technical detail in the following anymore). 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 mode with unconditional halting are denoted by $XEOtP_1(anti_{1,*}, max, u)$ and $XOtP_1(anti_{1,*}, max, u)$ for $X \in \{Ps, N\}$, respectively. We then get the following corollary:

Corollary 3. *For* $X \in \{Ps, N\}$, $XEOtP_1(anti_{1,*}, max, u) = XE0L$ *and* $XOtP_1(anti_{1,*}, max, u) = X0L$.

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

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 L systems:

Corollary 4. *For* $X \in \{Ps, N\}$ *and* $Y \in \{ncoo, anti_{1,*}\}$, $XED0L = XEDOtP_1(Y, max, u)$ *and* $XD0L = XDOtP_1(Y, max, u)$.

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 $PsEDOtP_1(ncoo, max, u) \subseteq PsED0L$ 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 $L(\Pi) = Ps_u(G) \in PsED0L$. 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.

Without extensions, using more than one cell in tissue P systems yields additional computational power: as is well known, the finite language $\{a, aa\}$ cannot be generated by a $0L$ system. On the other hand, every finite set of non-negative integers can be generated by a non-extended deterministic tissue P system with non-cooperative rules and unconditional halting:

Example 1. Consider the deterministic tissue P system

$$\Pi = (n, \{a\}, \{a\}, R, (a^{x_1}, \dots, a^{x_n}), n)$$

with $R = \{(a,i) \to (a,i+1) \mid 1 \leq i < n\}$. In the first $n-1$ computation steps, the contents of the first cells is shifted one cell further in the sequence of cells 1 to n; after these $n-1$ transition steps the contents of all cells $< n$ is empty and the computation stops. During the computation, at step i, $0 \leq i < n$, $a^{y_{i+1}}$ where $y_{i+1} = \sum_{j=0}^{i} x_{n-j}$, is found in the output cell n; hence, we obtain $N_u(\Pi) = \{y_i \mid 1 \leq i \leq n\}$.

As an immediate consequence of Corollary 2 and the preceding example, we obtain the following result:

Theorem 2. $NDOP_2(ncoo, max, u) - N0L \neq \emptyset$.

We conjecture that the families $NDOP_n(ncoo, max, u)$ form an infinite hierarchy with respect to the number of cells; a formal proof of this conjecture is left as a challenging task for future research.

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) L 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 transition modes as the sequential or the minimally parallel 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 L 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].

References

1. Dassow, J., Păun, G.: Regulated Rewriting in Formal Language Theory. Springer, Berlin (1989)
2. Freund, R., Păun, G., Pérez-Jiménez, M.J.: Tissue-like P systems with channel states. Theoretical Computer Sci. 330, 101–116 (2005)
3. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Efficient computation in real-valued P systems. In: The Sixth Brainstorming Workshop On Membrane Computing, Sevilla (2008)
4. Martín-Vide, C., Păun, G., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Sci. 296, 295–326 (2003)
5. Păun, G.: Computing with membranes. J. Computer and System Sci. 61, 108–143 (2000)
6. Păun, G. (ed.): Membrane Computing. An Introduction. Springer, Berlin (2002)
7. Rozenberg, G., Salomaa, A.: The Mathematical Theory of L Systems. Academic Press, New York (1980)
8. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer, Berlin (1997)
9. The P Systems Web Page, `http://ppage.psystems.eu`

Modeling 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], and Antoni Margalida[4]

[1] Dept. of Mathematics, University of Lleida
Av. Alcalde Rovira Roure, 191. 25198 Lleida, Spain
{mcardona,colomer}@matematica.udl.es
[2] Research Group on Natural Computing
Dept. of Computer Science and Artificial Intelligence, University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
marper@us.es
[3] Dept. of Animal Production, University of Lleida
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

Abstract. The Bearded Vulture (*Gypaetus barbatus*) is an endangered species in Europe that feeds almost exclusively on bone remains of wild and domestic ungulates. In this paper, we present a model of an ecosystem related to the Bearded Vulture in the Pyrenees (NE Spain), by using P systems. The evolution of six species is studied: the Bearded Vulture and five subfamilies of domestic and wild ungulates upon which the vulture feeds. P systems provide a high level computational modeling framework which integrates the structural and dynamic aspects of ecosystems in a comprehensive 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 experimentally validate the P system designed, we have constructed a simulator that allows us to analyze the evolution of the ecosystem under different initial conditions.

1 Introduction

Animal species are interconnected in a network in which some species depend on others in terms of feeding [10], [26]. Variations in biomass affect the composition of the population structures [24]. In mountain ecosystems, the presence of domestic animals has disrupted the traditional relationships between wild ungulates and their predators[6]. Animals located at the top of the ecological pyramid are susceptible to the presence and number of these domestic animals.

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 137–156, 2009.

The abandonment of dead animals in the mountains is a major source of food for necrophagous species [15]. This is the case for 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 species interact with the environment [13] is one aspect of conservation biology of great interest to managers and conservationists [2]. A widespread tool used in this area is the ecological model, which uses 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 modeling framework which integrates the structural and dynamic aspects of ecosystems in a comprehensive 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 a 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 experimentally validate of the P system designed we have constructed a simulator that allows us to analyze 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 behavior 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 modeled are introduced. The most outstanding aspects of each species are detailed as well as the interactions among them. In Section 3, a dynamic 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 Modeling the Ecosystem

The ecosystem to be modeled 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's Spanish population in 2007 (n = 102). See Figure 1 [15].

The ecosystem to be modeled 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.

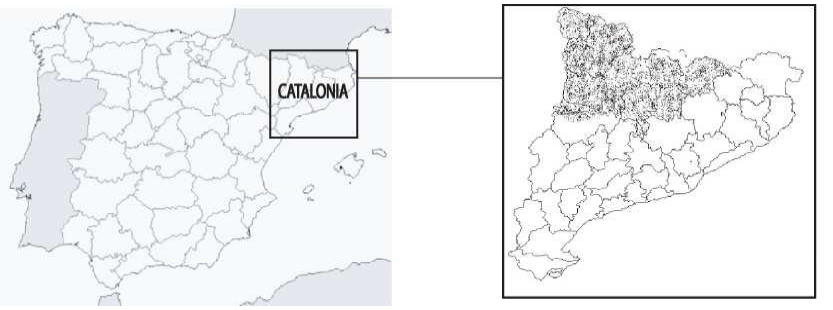


Fig. 1. Regional distribution of the Bearded Vulture in the Catalan Pyrenees

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 takes place from December to February and after 52-54 days of incubation and after about 120 days of chick-rearing, the chick abandons the nest between June and August [19]. Clutch size in this species is usually 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 remaining 33% includes the remains of small sized mammals (e.g., dogs, cats), large mammals (cows, horses), medium sized mammals (e.g., wild boars) and birds [15]. A pair of Bearded Vultures needs an average of 341 Kg of bones per year [17],[16].

During the dispersal period (from fledgling until the birds become territorial at 6 or 7 years), non-adult Bearded Vultures cover large distances surveying different areas. For example, the average surface covered by four young vultures 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 at 4-5%. The floating population principally remains in feeding stations situated in the central Pyrenees (Aragon).

The natural behavior of the five bovid species is similar because they are all herbivores and they all reach adult size at one year of age. In general, they reach sexual maturity within two years of 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 their remaining years. In spite of the great degree of similarity between these five species, important differences exist. For example, some are naturally occurring while others have been introduced by human populations0. It is essential to bear these differences in mind while defining a P system that can simulate the ecosystem in a reliable way.

Red Deer are appreciated 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 in the field. Hence the carcass is eaten by other species and the bone remains 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 remains of all of the dead animals of these species.

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 than their actual life expectancy in the field because upon a decrease in fertility rate at the age of eight, they are removed from the habitat. Most of the lambs are sold to market and so they are removed from the habitat in the first year of life. Only 20% to 30% of the lambs, mostly females, are left in the field and these are used to replace sheep that have died naturally and those older sheep that have been removed from the flock. The number of animals in the Catalan Pyrenees during the years 1994 and 2008 is shown in Table 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 the fact that the availability of food limits the feeding of the herbivores, and so the growth of vegetation is not modeled.

Taking all of 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 consumes as much as an adult. At this age the animal will have surpassed the critical early phase during which mortality rate is high;
- I_2: Age at which it begins 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 100 fertile females);
- I_6: Mortality ratio in first years, $age < I_1$ (this quantity is expressed in terms of percentage);
- I_7: Mortality ratio in adult animals, $age \geq I_1$ (this quantity is expressed in terms of percentage);
- I_8: Ratio of females in the population (this quantity is expressed in terms of percentage).

The required information about each species is shown in Table 2 (see the Appendix).

When an animal dies, the weight of the bones that it leaves behind is around 20% of its total weight. Table 3 (see Appendix) shows the average weight of each animal as well as the weight of bones left behind. In the case of Fallow Deer and Roe Deer, the value of the weight of bones is then multiplied by 0,2 (20%) which is the portion of bones from which the Bearded Vulture may benefit.

In the P system only Bearded Vultures older than 8 are considered, because 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 feed at these sites.

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 the web and published in 2000 [22]). Since then it has received important attention from the scientific community. Details can be found on the web page `http://ppage.psystems.eu/`

In short, one abstracts computing models from the structure and the functioning of living cells, as well as from the organization of cells 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 to 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 so). In this way, we obtain *transitions* from one configuration of the system to the next.

A *computation* of the system is a (finite or infinite) sequence of configurations such that each is obtained from the previous by a transition, and shows how the system is evolving. A computation that reaches a configuration in which 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 behavior 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*

$$\Pi = (\Gamma, \mu, w_1, \ldots, w_n, R, \{c_r\}_{r \in R}),$$

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 1. We also associate electrical charges with membranes from the set* $\{0, +\}$, *neutral and positive;*
- $w_1, \ldots, w_n$ *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 labeled 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 $(w_1, \ldots, w_n)$ is the initial configuration of the system.

The P system can pass from one configuration to another 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 of the following probabilistic P system of degree 2 with two electrical charges (neutral and positive):

$$\Pi = (\Gamma, \mu, w_1, w_2, R, \{c_r\}_{r \in R}),$$

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 states); it also contains the auxiliary symbol B, which represents 0.5 kg of bones, and C, which allows a change in the polarization of the membrane labeled by 2 at 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 consider two regions, $\mu = [\ [\ ]_2\ ]_1$ (neutral polarization will be omitted):

  - the skin region where the objects that represent animals evolve according to the rules of reproduction and mortality.
  - an inner membrane where the objects associated with animals evolve according to the feeding rules.
- In w_1 and w_2, we specify the initial number of objects present in each region (encoding the initial population and the initial food);
  - $w_1 = \{X_{ij}^{q_{ij}} \mid 1 \leq i \leq 7,\ 0 \leq j \leq k_{i,4}\}$, where the multiplicity q_{ij} indicates the number of animals, of species i whose age is j that are initially present in the ecosystem;
  - $w_2 = \{C\ B^{\alpha}\}$, 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 of Bearded Vultures, the finality of constant factor 1.10 is to guarantee enough food for 10% population growth. At present, the population growth is estimated an average 4%, but this value can reach higher values. Thus, to avoid problems related with the underestimation of this value the first year we estimated the population growth (overestimated) at 10%. The constant value 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:
    * $r_0 \equiv [X_{ij} \xrightarrow{(1-k_{i,13})\cdot(1-k_{i,15})} Y_{ij}]_1,\ 1 \leq i \leq 7, k_{i,2} \leq j \leq k_{i,4}.$

    Adult females that reproduce:
    * $r_1 \equiv [X_{ij} \xrightarrow{k_{i,5}\cdot k_{i,13}\cdot(1-k_{i,15})} Y_{ij}Y_{i0}]_1,\ 1 \leq i \leq 7,\ k_{i,2} \leq j < k_{i,3}.$

    Fertile adult females that do not reproduce:
    * $r_2 \equiv [X_{ij} \xrightarrow{(1-k_{i,5})\cdot k_{i,13}\cdot(1-k_{i,15})} Y_{ij}]_1,\ 1 \leq i \leq 7,\ k_{i,2} \leq j < k_{i,3}.$

    Not fertile adult females:
    * $r_3 \equiv [X_{ij} \xrightarrow{k_{i,13}\cdot(1-k_{i,15})} Y_{ij}]_1,\ 1 \leq i \leq 7,\ k_{i,3} \leq j \leq k_{i,4}.$

    Young animals that do not reproduce:
    * $r_4 \equiv [X_{ij} \xrightarrow{1-k_{i,15}} Y_{ij}]_1,\ 1 \leq i \leq 7,\ 0 \leq j < k_{i,2}.$
  - Growth rules.
    * $r_5 \equiv [X_{ij} \xrightarrow{(k_{i,6}+k_{i,10})\cdot k_{i,15}} Y_{ik_{i,2}}Y_{ij}]_1,\ 1 \leq i \leq 7,\ k_{i,2} \leq j < k_{i,4}.$
    * $r_6 \equiv [X_{ij} \xrightarrow{(1-k_{i,6}-k_{i,10})\cdot k_{i,15}} Y_{ij}]_1,\ 1 \leq i \leq 7,\ k_{i,2} \leq j < k_{i,4}.$
    * $r_7 \equiv [X_{ij} \xrightarrow{k_{i,6}\cdot k_{i,15}} Y_{ik_{i,2}}Y_{ij}]_1,\ 1 \leq i \leq 7,\ j = k_{i,4}.$
    * $r_8 \equiv [X_{ij} \xrightarrow{(1-k_{i,6})\cdot k_{i,15}} Y_{ij}]_1,\ 1 \leq i \leq 7,\ j = k_{i,4}.$

- Mortality rules.
  - Young animals

  Those which survive:
  * $r_9 \equiv Y_{ij}[\ ]_2 \xrightarrow{1-k_{i,7}-k_{i,8}} [Z_{ij}]_2,\ 1 \leq i \leq 7,\ 0 \leq j < k_{i,1}.$

  Those which die:
  * $r_{10} \equiv Y_{ij}[\ ]_2 \xrightarrow{k_{i,8}} [B^{k_{i,11}}]_2,\ 1 \leq i \leq 7,\ 0 \leq j < k_{i,1}.$

  Those which are retired from the ecosystem:
  * $r_{11} \equiv [Y_{ij} \xrightarrow{k_{i,7}} \lambda]_1,\ 1 \leq i \leq 7,\ 0 \leq j < k_{i,1}.$
  - Adult animals that don't arrive at an average life expectancy .

  Those which survive:
  * $r_{12} \equiv Y_{ij}[\ ]_2 \xrightarrow{1-k_{i,10}} [Z_{ij}]_2,\ 1 \leq i \leq 7,\ k_{i,1} \leq j < k_{i,4}.$

  Those which die:
  * $r_{13} \equiv Y_{ij}[\ ]_2 \xrightarrow{k_{i,10}} [B^{k_{i,12}}]_2,\ 1 \leq i \leq 7,\ k_{i,1} \leq j < k_{i,4}.$
  - Animals that arrive at an average life expectancy:

  Those which growth population depend on the fertility ratio and die in the ecosystem:
  * $r_{14} \equiv Y_{ij}[\ ]_2 \xrightarrow{(1-k_{i,15})\cdot(k_{i,9}+(1-k_{i,9})\cdot k_{i,10})} [B^{k_{i,12}}]_2,\ 1 \leq i \leq 7,\ j = k_{i,4}.$

  Those which growth population depend on the fertility ratio and are retired of the ecosystem:
  * $r_{15} \equiv [Y_{ij} \xrightarrow{(1-k_{i,15})\cdot(1-k_{i,9})\cdot(1-k_{i,10})} \lambda]_1,\ 1 \leq i \leq 7,\ j = k_{i,4}.$

  Those which growth population not depend on the fertility ratio:
  * $r_{16} \equiv Y_{ij}[\ ]_2 \xrightarrow{k_{i,15}} [Z_{ik_{i,2}}]_2,\ 1 \leq i \leq 7,\ j = k_{i,4}.$
- Feeding rules.
  * $r_{17} \equiv [Z_{ij}B^{k_{i,14}}]_2 \rightarrow X_{ij+1}[\ ]_2^+,\ 1 \leq i \leq 7,\ 0 \leq j \leq k_{i,4}.$
- Balance rules. The propose of this rules is to make a balance at the end of the year. It is to say the leftover food not served for the next year, so it is necessary eliminate, and if the amount of food not is enough some animals die.

  Elimination of remaining bones:
  * $r_{18} \equiv [B]_2^+ \rightarrow [\ ]_2.$

  Adult animals that die because they have not enough food:
  * $r_{19} \equiv [Z_{ij}]_2^+ \rightarrow [B^{k_{i,12}}]_2,\ 1 \leq i \leq 7,\ k_{i,1} \leq j \leq k_{i,4}$

  Young animals that die because they have not enough food:
  * $r_{20} \equiv [Z_{ij}]_2^+ \rightarrow [B^{k_{i,11}}]_2,\ 1 \leq i \leq 7,\ j < k_{i,1}$

  If the food is equal to the necessary the object C allow to change the polarization.
  * $r_{21} \equiv [C]_2^+ \rightarrow [C]_2.$

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 consumes food as an adult does, and at which, if the animal dies, the amount of biomass it leaves behind is similar to the total 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 begins to be fertile.
- $k_{i,3}$: Age at which it stops being fertile.
- $k_{i,4}$: Average life expectancy in the ecosystem.
- $k_{i,5}$: Fertility ratio (number of descendants by fertile females).
- $k_{i,6}$: Population growth (this quantity is expressed in terms of 1).
- $k_{i,7}$: Animals retired from the ecosystem in the first years, $age < k_{i,1}$ (this quantity is expressed in terms of 1).
- $k_{i,8}$: Natural mortality ratio in first years, $age < k_{i,1}$ (this quantity is expressed in terms of 1).
- $k_{i,9}$:0 if the live animals are retired at age $k_{i,4}$, in other cases, the value is 1.
- $k_{i,10}$: Mortality ratio in adult animals, $age \geq k_{i,1}$ (this quantity is expressed in terms of 1).
- $k_{i,11}$: Amount of bones from young animals, $age < k_{i,1}$.
- $k_{i,12}$: Amount of bones from adult animals, $age \geq k_{i,1}$.
- $k_{i,13}$: Proportion of females in the population (this quantity is expressed in terms of 1).
- $k_{i,14}$: Amount of food necessary per year and breeding pair (1 unit is equal to 0.5 kg of bones).
- $k_{i,15}$: Equal to 0 when the species experience natural growth (animals that remain in the same territory throughout their lives) and is equal to 1 when animals are nomadic (the Bearded Vulture moves from one place to another until it is 6–7 years old, at which point it remains in one location).

Values for each species are shown in Table 4 (see Appendix). Most values in that table are equal to those in Table 2 (see Appendix), but it is necessary to

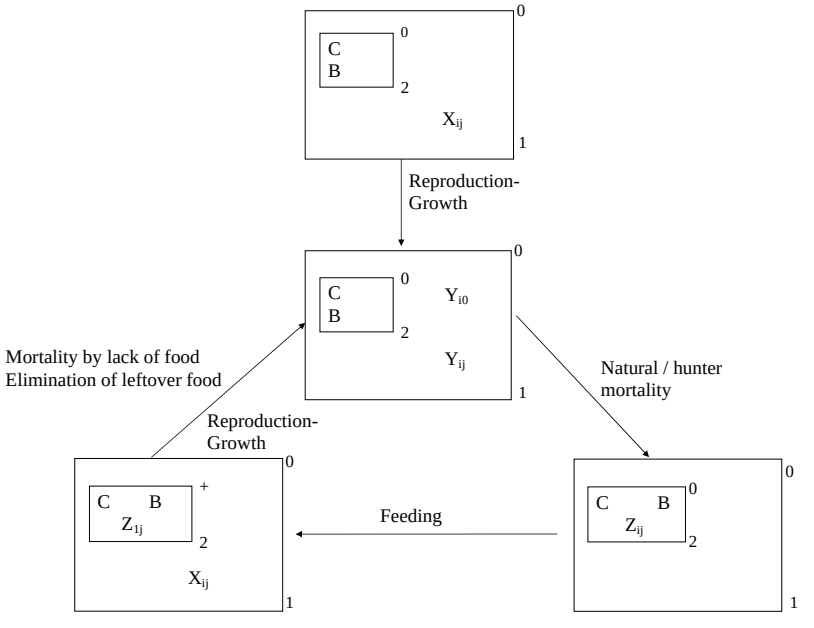


Fig. 2. Structure of the P system running

remark on values $k_{4,10}$ and $k_{1,14}$. Value $k_{4,10}$ is obtained by adding 6% of natural mortality to 30% of animals killed by hunters. Value $k_{1,14}$ is 67% of 682 units $(341 \cdot 2)$ which is the food the Bearded Vulture obtains from the other five species of ungulates modeled 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 due to lack of food takes place. These stages are depicted in Figure 2.

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 that 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 a 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.

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 within 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 an experimental way. 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 provides 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 ceases to play 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 the 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, takes place at the same time for all of the species considered in this study. It will be necessary to verify whether 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 the unit of time, so it has been necessary to discretize feeding and mortality variables.

As shown in Table 1 (see Appendix), data on 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 5 (see Appendix), and another for the year 2008 which has been used to study the robustness of the P system, shown in Table 6 (see Appendix).

5.1 Robustness

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

According to the design of the P system, reproduction rules have a higher priority than mortality rules. Subsequently, 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 to variable Y in the initial multiset $\mathcal{M}_1$.

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

In Figure 3, solid lines and dashed 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 behavior is similar in both cases, it can be deduced that our model is robust with regard to the properties considered.

5.2 Experimental Validation

Let us suppose that we are studying a phenomenon for which we have (a sufficient amount of) data experimentally obtained (in the 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, 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.

It should be noted that Table 1 (see Appendix) shows those data experimentally obtained corresponding to the years from 1994 to 2008 (the input being the number of animals in 1994) covering a period of 14 years. We have run our simulator 10 times as it supposes a reduction of 70% of the deviation.

Table 7 (see Appendix) and Figure 4 show the difference between the average number of animal species obtained with the simulator compared with the censuses

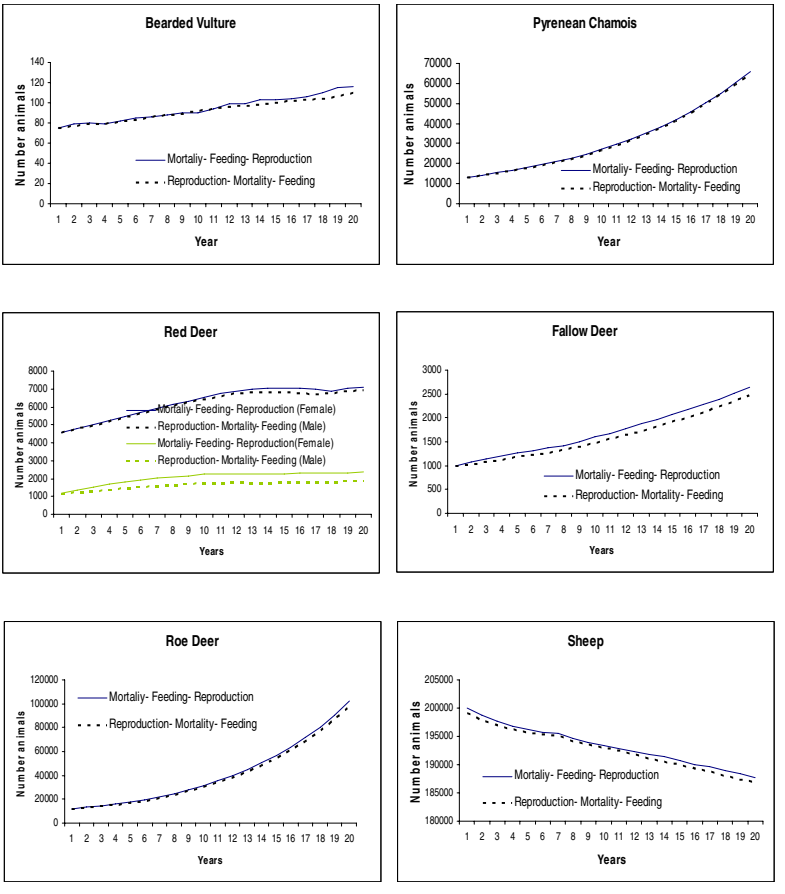


Fig. 3. Robustness of the ecosystem

estimated for 2008. In 2004, the Pyrenean Chamois species was affected by a disease which led to a decrease in the population to 10000 animals. In the third column of that table, the evolution of the P system is shown without taking into account this piece of information, while in the fourth column it has been considered.

Figure 5 shows the deviation and the coefficient of variation (percentage of deviation with respect to the average) for every year of the simulated data. The deviation increases as we move away from the initial year. Therefore, the noise is greater when we make predictions in the long term. The coefficient of variation obtained in the species studies over 14 years does not exceed the 14% for the Bearded Vulture and the 5% for the rest of the species. In the case of the Bearded Vulture, the value is higher than the others due to the low number of breeding pairs of the Bearded Vulture.

The proposed P system can be considered a good model for the study of the evolution of an ecosystem. Variations noticed among the available data regarding

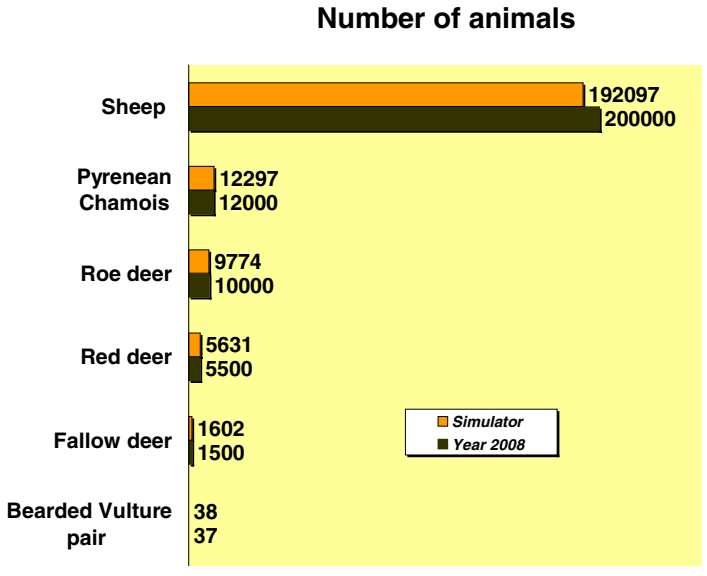


Fig. 4. Average number of animals obtained with the simulator in comparison with the censuses estimated for 2008

the number of animals of each species from 1994 to 2008 (see Table 1) (see Appendix) are of almost no importance if we take into account that these data are taken from estimated censuses and are never exact. Under the same conditions as a starting point, the ecosystem has a certain behavior 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]. This is why the population of some of the species such as Roe Deer, Fallow Deer and Chamois may grow exponentially reaching values which cannot be obtained 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 begins that imposes a drastic decrease of the population to 1000 individuals. Hence, if density are not taken into account, it may not be suitable for the study of ecosystem dynamics in the long–term.

Neither ungulates' feeding behavior nor their population density have been taken into account. This implies an exponential growth of ungulate species constituting the basic food source for the Bearded Vulture. Consequently, there is a continuous growth in the number of pairs of Bearded Vultures. According to some researchers [14], the estimated maximum number of pairs of Bearded Vultures within the area under study is about fifty. Higher numbers of pairs

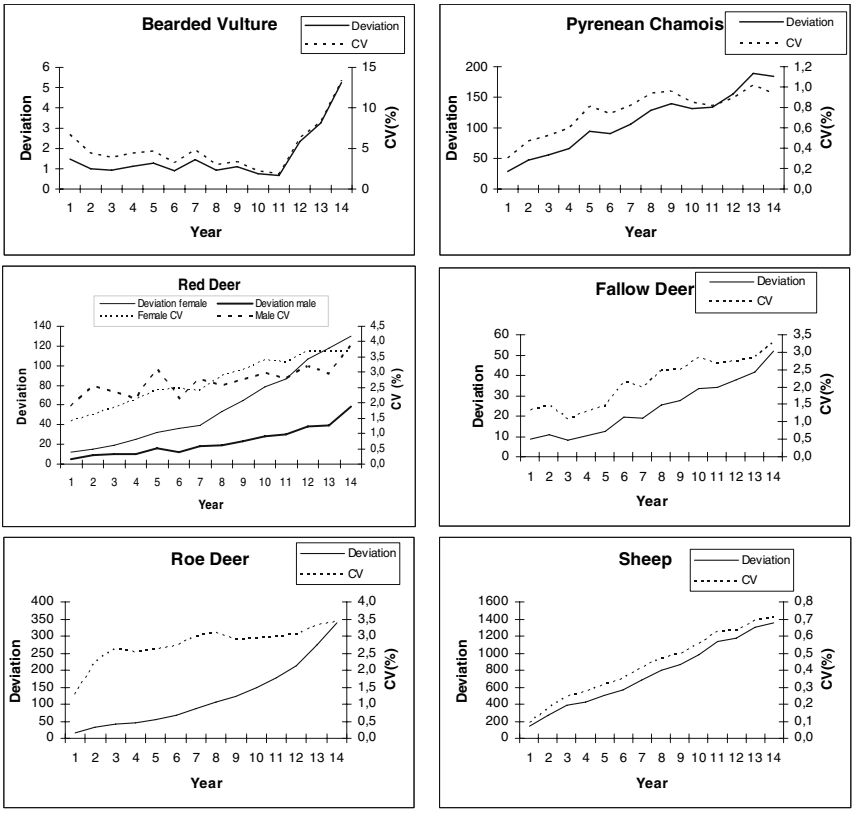


Fig. 5. Deviation and Coefficient of Variation

would lead to competition and a subsequent decrease in the population down to values that the ecosystem can handle.

6 Conclusions and Future Work

A probabilistic P system which models an ecosystem related to the Bearded Vulture 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 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 future work, this factor and other parameters (i.e., the amount of food of the herbivorous species, the climatic changes in the ecosystem, etc.) should be considered.

In order to obtain a model which allows 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 in which the ungulates may feed.
- Amount of food eaten daily by each of the ungulate species considering their age.

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

In future work, we will also try to model interactions between neighboring ecosystems.

Acknowledgements. M.J. Pérez–Jiménez acknowledges 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 Excellence 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.

References

1. Antor, R.J., Margalida, A., Frey, H., Heredia, R., Lorente, L., Sesé, J.A.: Age of first breeding in wild and captive populations of Bearded Vultures (Gypaetus barbatus). Acta Ornithologica 42, 114–118 (2007)
2. Begon, M., Harper, J.L., Townsend, C.R.: Ecology: Individuals, Populations and Communities. Blackwell Scientific Publications Inc., Oxford (1988)
3. Braza, F., San José, C., Blom, A., Cases, V., García, J.E.: Population parameters of fallow deer at Doñana National Park (SW Spain). Acta Theriol. 35, 277–288 (1990)
4. Brown, C.J.: Population dynamics of the bearded vulture Gypaetus barbatus in southern Africa. African J. Ecology 35, 53–63 (1997)
5. Cardona, M., Colomer, M.A., Pérez-Jímenez, M.J., Zaragoza, A.: Handling markov chains with membrane computing. In: Calude, C.S., Dinneen, M.J., Păun, G., Rozenberg, G., Stepney, S. (eds.) UC 2006. LNCS, vol. 4135, pp. 72–85. Springer, Heidelberg (2006)
6. Chocarro, C., Fanlo, R., Fillat, F., Marín, P.: Historical evolution of natural resource use in the central Pyrenees of Spain. Mountain Research And Development 10, 257–265 (1990)
7. Clutton-Brock, T., Guinness, F.E., Albon, S.D.: Red deer: Behavior and Ecology of Two Sexes. Edinburgh University Press, Edinburgh (1982)
8. García-González, R., Herrero, J., Hidalgo, R.: Estimación puntual de diversos parámetros poblacionales y distributivos del sarrio en el Pirineo Occidental. Pirineos 35, 53–63 (1985)

9. Garin, I., Herrero, J.: Distribution, abundance and demographic parameters of the Pyrenean Chamois (Rupicapra p. pyrenaica) in Navarre, Western Pyreness. Mammalia 61, 55–63 (1997)
10. Harvey, P.H., Purvis, A.: 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, pp. 232–247. Blackwell Science Publications, Oxford (1999)
11. Heredia, R.: Status y distribución del quebrantahuesos en España y diagnóstico de la situación de la población en la UE. In: Margalida, A., Heredia, R. (eds.) Biología de la conservación del quebrantahuesos Gypaetus barbatus en España, Organismo Autónomo Parques Nacionales, Madrid (2005)
12. Krüger, O.: Long-term demographic analysis in goshawk. Accipiter gentilis: the role of density dependence and stochasticity. Oecologia 152, 459–471 (2008)
13. Margalef, R.: Ecologa. Universidad Nacional de Educación a Distancia, Madrid (1977)
14. Margalida, A., Donázar, J.A., Bustamante, J., Hernández, F.J., Romero-Pujante, M.: 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, 242–249 (2008)
15. Margalida, A., Garcí, D., Cortés-Avizanda, A.: 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, 189–200 (2007)
16. Margalida, A., Mañosa, S., Bertran, J., Garcıa, D.: Biases in studying the diet of the Bearded Vulture. The Journal of Wildlife Management 71, 1621–1625 (2006)
17. Margalida, A., Bertran, J., Boudet, J.: Assessing the diet of nestling Bearded Vultures: a comparison between direct observation methods. Journal of Field Ornithology 76, 40–45 (2005)
18. Margalida, A., Bertran, J., Boudet, J., Heredia, R.: Hatching asynchrony, sibling aggression and cannibalism in the Bearded Vulture (Gypaetus barbatus). Ibis 146, 386–393 (2004)
19. Margalida, A., García, D., Bertran, J., Heredia, R.: Breeding biology and success of the Bearded Vulture Gypaetus barbatus in the eastern Pyrenees. Ibis 145, 244–252 (2003)
20. Mateos-Quesada, P., Carranza, J.: Reproductive patterns of roe deer in central Spain. Etologa 8, 9–12 (2000)
21. McCallum, H.: Population parameters: estimation for ecological models. Blackwell Science Publications, Oxford (2000)
22. Păun, G.: Computing with membranes. J. Computer and System Sci. 61, 108–143 (2000)
23. Sunyer, C.: El periodo de emancipación en el Quebrantahuesos: consideraciones sobre su conservación. In: Heredia, R., Heredia, B. (eds.) El Quebrantahuesos (Gypaetus barbatus) en los Pirineos, Colección Técnica, Madrid, pp. 47–65 (1991)
24. Tilman, D.: Resource Competition and Community Structure. Princeton University Press, Princeton (1982)
25. Torres, R.: Conservación de recursos genéticos ovinos en la raza Xisqueta: caracterización estructural, racial y gestión de la diversidad en programas “in situ”. Ph D Thesis. Universitat Autónoma de Barcelona, Barcelona (2006)
26. Watson, A.: Animal Populations in Relation to Their Food Resources. Blackwell Scientific Publications, Oxford (1970)

Appendix

Table 1. Number of animals presents in the Catalan Pyrenees between 1994–1998

Species	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 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 3. Descriptive variables used to model the ecosystem

Species	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	50	5.75	-	-	-
Pyrenean 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 4. Constants used in the P system based model

Species	i	$k_{i,1}$	$k_{i,2}$	$k_{i,3}$	$k_{i,4}$	$k_{i,5}$	$k_{i,6}$	$k_{i,7}$	$k_{i,8}$	$k_{i,9}$	$k_{i,10}$	$k_{i,11}$	$k_{i,12}$	$k_{i,13}$	$k_{i,14}$	$k_{i,15}$
Bearded Vulture	1	1	8	20	21	0.38	0.04	0	0.06	1	0.12	0	0	0.50	460	1
Pyrenean Chamois	2	1	2	18	18	0.75	-	0	0.60	1	0.06	6	12	0.55	-	0
Red Deer Female	3	1	2	17	17	0.75	-	0	0.34	1	0.06	15	30	1.00	-	0
Red Deer Male	4	1	2	-	20	-	-	0	0.34	1	0.36	24	48	0	-	0
Fallow Deer	5	1	2	12	12	0.55	-	0	0.50	1	0.06	2	4	0.75	-	0
Roe Deer	6	1	1	10	10	1.00	-	0	0.58	1	0.06	1	2	0.67	-	0
Sheep	7	1	2	8	8	0.75	-	0.57	0.15	0	0.03	7	14	0.96	-	0

Table 5. Estimation of number of animals per age in 1994

Age	Bearded V.	Chamois	Red d. female	Red d. 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. Estimation of number of animals per age in 2008

Age	Bearded V.	Chamois	Red d. female	Red d. 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 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 and Vincenzo Manca

Verona University, Computer Science Department
Strada Le Grazie 15, 37134 Verona, Italy
{alberto.castellini,vincenzo.manca}@univr.it

Abstract. In this work the formalism of metabolic P systems is employed as a basis of a new computational framework for modeling biological networks. The proposed software is a virtual laboratory, called MetaPlab, which supports the synthesis of metabolic P systems by means of an extensible plugin-based architecture. The Java implementation of the software is outlined and a specific plugin at work is described to highlight the internal functioning of the whole 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 [13,26]. In this way 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*. The computational power of many P system classes have been proved so far [24]. P systems seem especially apt to model biological systems but their original mathematical setting is too abstract for expressing real biological phenomena.

Metabolic P systems, also called *MP systems*, are a class of P systems proved to be effective and successful for modeling biological phenomena related to metabolism (matter transformation, assimilation and expulsion in living organisms). They were conceived in [20] and subsequently extended in many works [4,5,18,15,16,19]. 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].

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 157–168, 2009.

Equivalence results have been proved, in [10] and [9,8], 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 [4,20], the SIR (susceptible-infected-recovered) epidemic [4], the protein kinase C activation [5], the circadian rhythms, the mitotic cycles in early amphibian embryos [19], a Pseudomonas quorum sensing model [1,6] and the *lac* operon gene regulatory mechanism in glycolytic pathway [9]. In order to simulate MP systems we developed a first Java computational tool called *MPsim* [3]. The current release of the software, available at [28], 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, within the MP systems framework, by the *Log-gain* theory [17,18] which roots in allometric 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 about substance quantities and parameter values 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 [21] it provided a model of a photosynthesis 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 allometric 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 into 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 has been released [29] 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* [12], 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 as well 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 parameter estimation and model analysis. The new computational framework we propose in the following is based on an extensible set of *plugins*, namely Java tools for solving specific tasks relevant in the framework of MP systems, such as regulation function discovery, 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 developed to manage these problems in a systematic way, and finally, a plugin for computing MP systems dynamics is presented in Section 4 with a complete functioning description.

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 [18,19].

Definition 1 (MP system). *An MP system is a discrete dynamical system specified by a construct [17]*

$$M = (X, R, V, Q, \Phi, \nu, \mu, \tau, q_0, \delta)$$

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 $\alpha_r \to \beta_r$ 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:*

$$\delta(i+1)_{|X} = \mathbb{A} \times U(\delta(i)) + \delta(i)_{|X} \tag{1}$$

$$\delta(i+1)_{|V} = (h_v(i+1) \mid v \in V) \tag{2}$$

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 [19], 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 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 chemo-physical 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 (1) and (2).

The *dynamics computation* task, just defined, is only one of several biologically inspired mathematical problems which can be tackled by MP systems. Table 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 $U(\delta(0)), \ldots, U(\delta(i-1))$ related to observed dynamics $\delta_{<i}$ of substances and parameters. A mathematical theory

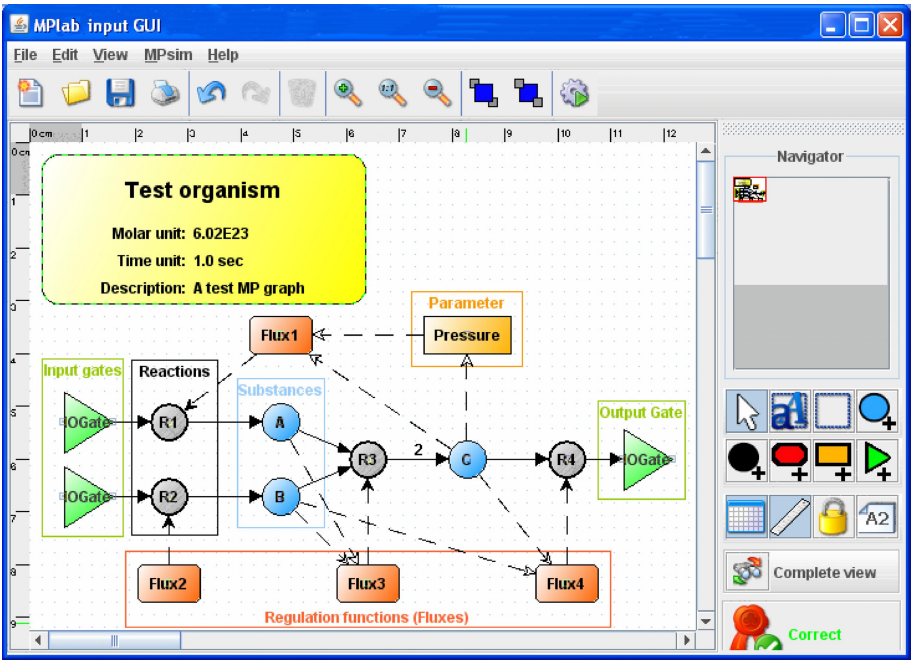


Fig. 1. An MP graph visualized by a graphical user interface of MetaPlab. Frame labels point out MP system elements in the MP graph representation. Substances, reactions and parameters describe the system stoichiometry, while fluxes express the system regulation.

Table 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.

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.

for solving this problem, called Log-gain theory, has been proposed in [17,18], 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 [21] as well as by evolutionary computing techniques, such as genetic programming [14] and neural networks [7].

The last problem listed in Table 1 concerns the *dynamics analysis* of an MP system, 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, 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 extensible set of plugins, each dedicated to a specific task.

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 1. Figure 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 the previous software MPsim, which only coped with the MP systems simulation. The new extensible 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 2. A technical description of the whole architecture is available in the MetaPlab User Guide [29].

MP graphs. The leftmost layer of Figure 2 contains the MetaPlab input GUI, also depicted in Figure 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 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 experimental 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.

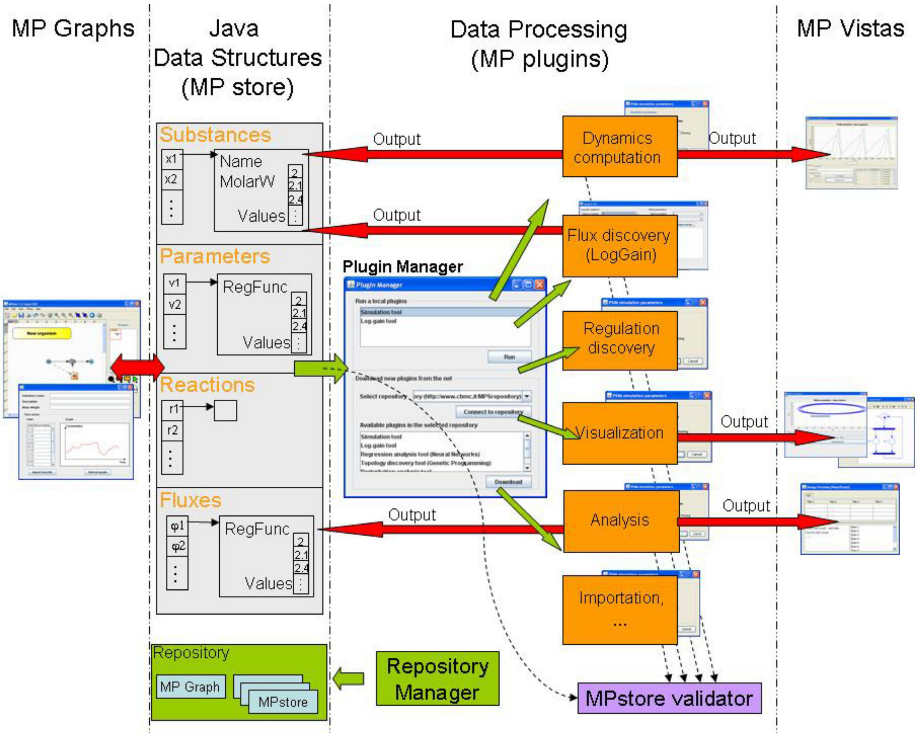


Fig. 2. MetaPlab framework

MP store data structure. The second layer of Figure 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 *i)* the substance name x, *ii)* its molar weight $\mu(x)$ and *iii)* 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, indeed 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 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. In fact, 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 2 represents the core of the new architecture. It is a plugin-based module coping with the MP systems data processing. This layer is composed by *i)* an extensible 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 models 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 **outputs** from plugins 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 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 the MetaPlab framework. In this way, the compiled (*.jar*) file of the plugin must be placed into a specific folder, called *plugin directory*, in order to be automatically recognized and loaded by the Plugin Manager.

Figure 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 will enable the uploading of new plugins from suitable repositories. Due to their intrinsic open and easy structure, MP plugins can be implemented just 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, in order to encourage 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. The MetaPlab website [29] supports the download of new plugins and it provides a complete software documentation.

MP vistas. The fourth level of the architecture deals with the graphical representation of MP structures and MP dynamics. These views support the analysis of specific features of an MP system. A vista can show, for instance, substance

Fig. 3. 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.

and parameter time-series charts together, plot phase diagrams or visualize statistical indexes.

Auxiliary modules. Two further modules are displayed in the bottom of Figure 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 describe the functioning of a real plugin in order to highlight the main mechanisms of the plugin-based framework described so far. The plugin we propose is a *simulator* which computes the dynamics of an MP system by applying the recurrent equations (1) and (2) of Definition 1. We remark that the plugin is simply a rearrangement of the stand-alone software MPsim. 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 requirements are published in the MetaPlab Developer Guide [29].

Plugin functioning. At the beginning of the modeling process, we define an MP system by dragging substance, parameter and reaction nodes from the right

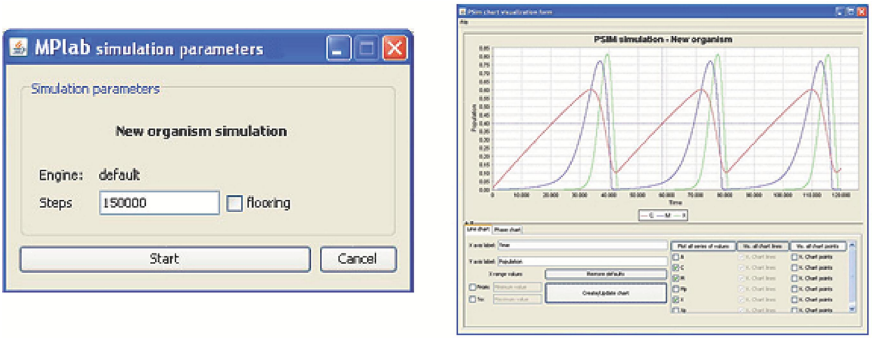


Fig. 4. On the left: input GUI of the dynamics computation plugin. **On the right:** output GUI of the dynamics computation plugin (vista).

toolbar to the central white panel of the input GUI (Figure 1). Then, we draw stoichiometric arcs, and we define both regulation functions and initial conditions by filling in the fields of suitable pop-up windows. For example, let us imagine to define an MP graph for a typical metabolic process, as the mitotic oscillator formerly simulated by the stand-alone tool in [3].

After this input stage we open the Plugin Manager (Figure 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 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 into an MP store data structure and the MP graph displayed by the input GUI updated with the new data. Furthermore, the dynamics is plotted by the plugin output interface (vista), depicted on the right of Figure 4, which shows the typical mitotic oscillations [11].

We finally remark that the data computed by this plugin 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.

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 an extensible plugin-based platform, namely 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. At present we are ready to add a new plugin,

based on the Log-gain theory [17], which compute the fluxes of a given MP system, from a temporal series of observed states. We also plan to develop other plugins based on traditional regression techniques, neural networks and genetic programming, for obtaining flux maps from fluxes.

Other important facilities 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/exportation from/to 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.

References

1. Bernardini, F., Gheorghe, M., Krasnogor, N.: Quorum sensing P systems. Theoretical Computer Sci. 371, 20–33 (2007)
2. Bernardini, F., Manca, V.: Dynamical aspects of P systems. BioSystems 70, 85–93 (2003)
3. Bianco, L., Castellini, A.: Psim: a computational platform for metabolic P systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 1–20. Springer, Heidelberg (2007)
4. Bianco, L., Fontana, F., Franco, G., Manca, V.: P systems for biological dynamics. In: Ciobanu, G., et al. (eds.) Applications of Membrane Computing, pp. 81–126. Springer, Heidelberg (2006)
5. Bianco, L., Fontana, F., Manca, V.: P systems with reaction maps. Intern. J. Foundations of Computer Sci. 17, 27–48 (2006)
6. Bianco, L., Pescini, D., Siepmann, P., Krasnogor, N., Romero-Campero, F.J., Gheorghe, M.: Towards a P systems Pseudomonas quorum sensing model. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 197–214. Springer, Heidelberg (2006)
7. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press, Oxford (1995)
8. Castellini, A., Franco, G., Manca, V.: Hybrid functional Petri nets as MP systems (submitted, 2008)
9. Castellini, A., Franco, G., Manca, V.: Toward a representation of hybrid functional Petri nets by MP systems. In: Proc. 2nd Intern. Workshop on Natural Computing, IWNC, Nagoya University, Japan (2007)
10. Fontana, F., Manca, V.: Discrete solutions to differential equations by metabolic P systems. Theoretical Computer Sci. 372, 165–182 (2007)
11. Goldbeter, A.: A minimal cascade model for the mitotic oscillator involving cyclin and cdc2 kinase. PNAS 88, 9107–9111 (1991)
12. Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P., Kummer, U.: COPASI a COmplex PAthway SImulator. Bioinformatics 22, 3067–3074 (2006)
13. Kitano, H.: Computational systems biology. Nature 420, 206–210 (2002)
14. Koza, J.R., Keane, M.A., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G.: Genetic Programming IV: Routine Human-Competitive Machine Intelligence (Genetic Programming). Springer, Heidelberg (2003)

15. Manca, V.: Metabolic P systems for biochemical dynamics. Progress in Natural Sciences 17, 384–391 (2007)
16. Manca, V.: Discrete simulations of biochemical dynamics. In: Garzon, M.H., Yan, H. (eds.) DNA 2007. LNCS, vol. 4848, pp. 231–235. Springer, Heidelberg (2008)
17. Manca, V.: Log-gain principles for metabolic P systems (submitted, 2008)
18. Manca, V.: The metabolic algorithm. Principles and applications. Theoretical Computer Sci. 404, 142–157 (2008)
19. Manca, V., Bianco, L.: Biological networks in metabolic P systems. BioSystems 91, 489–498 (2008)
20. Manca, V., Bianco, L., Fontana, F.: Evolutions and oscillations of P systems: Applications to biochemical phenomena. In: Mauri, G., Păun, G., Jesús Pérez-Jímenez, M., Rozenberg, G., Salomaa, A. (eds.) WMC 2004. LNCS, vol. 3365, pp. 63–84. Springer, Heidelberg (2005)
21. Manca, V., Pagliarini, R., Zorzan, S.: Toward an MP model of non photochemical quenching. In: Pre-Proc. 9-th Workshop on Membrane Computing, Edinburgh, UK (2008)
22. Nagasaki, M., Doi, A., Matsuno, H., Miyano, S.: Genomic object net: I. A platform for modelling and simulating biopathways. Applied Bioinformatics 2, 181–184 (2004)
23. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61, 108–143 (2000)
24. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
25. Romero-Campero, F.J., Cao, H., Camera, M., Krasnogor, N.: Structure and parameter estimation for cell systems biology models. In: Proc. Genetic and Evolutionary Computation Conference, GECCO 2008. ACM Publisher, New York (2008)
26. Voit, E., Neves, A.R., Santos, H.: The intricate side of systems biology. PNAS 103, 9452–9457 (2006)
27. von Bertalanffy, L.: General Systems Theory: Foundations, Developments, Applications. George Braziller Inc. (1967)
28. Center for BioMedival Computing web site, `http://wwwcbmc.it`
29. MetaPlab website, `http://mplab.sci.univr.it`

Usefulness States in New P System Communication Architectures

Juan Alberto de Frutos, Fernando Arroyo, and Alberto Arteta

Dpto. de Lenguajes, Proyectos y Sistemas Informáticos
Escuela Unversitaria de Informática - Universidad Politécnica de Madrid
Crta. de Valencia Km. 7 - 28031 Madrid - Spain
{jafrutos,farroyo,aarteta}@eui.upm.es

Abstract. 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 to 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 semantics of the P systems, in several distributed architectures. It will be shown 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 will be based on usefulness states, presented in a previous work, which allow each membrane of the system to know the set of membranes with which communication is possible at any time.

1 Introduction

Membrane computing [6] is a new branch of natural computing, inspired by 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 demonstrated to be equivalent to the Turing machine. Strictly speaking from an implementation point of view and considering only the simplest model transition P systems), there are several challenges for researchers in order to get real implementations of such systems. Today, one of the most interesting is to solve the communication 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 to the same processor, in order to reduce the number of

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 169–186, 2009.

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 total time.

The goal of this paper is to fit usefulness states into communication 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 be considered the required information for objects to reach their respective target membranes. This information is based on the usefulness state concept.

2 Related Works

In what follows, 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, in [8] an architecture named "partially parallel evolution with partially parallel communication" is presented. This architecture is based on the following ideas:

1. Membrane distribution. Several membranes are placed at each processor which will evolve, at worst, sequentially. Then, there are two kinds of communications: (i) internal communications between membranes allocated at the same processor, with negligible communication time due to the use of shared memory techniques, and (ii) 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 to 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 1 shows an example of a membrane structure for a transition P system and its distribution in an architecture with four processors.

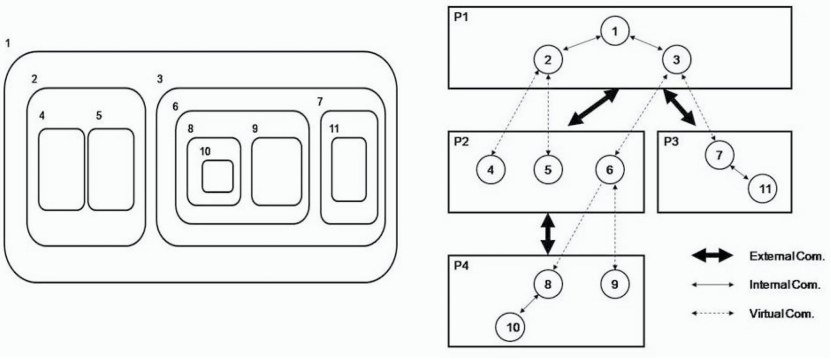


Fig. 1. Membrane distribution in processors

4. Token passing in communications in order to prevent 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 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 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 improvement 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 a result, a better evolution time of the system is obtained.

2.2 Usefulness States

The usefulness state concept for membranes of a P system was introduced in [5]. This state allows to any membrane to know the set of child 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 it is shown in [5]. One usefulness state in a membrane represents a valid context for that membrane, that is, a

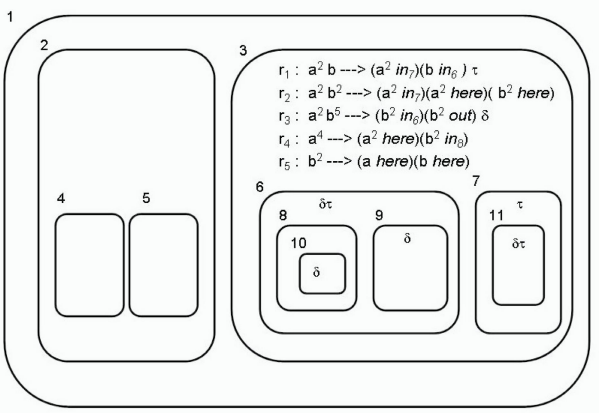


Fig. 2. Dissolving and inhibiting capabilities in membranes

context that can be reached after an evolution step. As the membrane context can change dynamically, transitions among states are also defined in [5].

From a given usefulness state we can obtain the set of useful rules. A rule is useful in an evolution step if all its targets are adjacent, not dissolved and not inhibited, hence the communication is feasible.

Figure 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 the communication through membranes 6, 7, and 11. Usefulness states for membrane 3 are depicted in table 1, together with their contexts and useful rules.

Tables defining transitions 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.

Table 1. Usefulness states for membrane 3

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

From an implementation point of view, problems arise when membranes have a high 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 usefulness state is encoded depending on its context, hence transitions are carried out directly in the code. Two definitions are introduced:

Total context for membrane j. This is the set of all membranes that eventually can become children of membrane j. Therefore, all contexts are included in the total context:

$$TC(j) = Child\_Of(j) \bigcup_{j_k \in Child\_D(j)} TC(j_k), \tag{1}$$

where $Child\_Of(j)$ is the set of all membrane j children in the initial structure, and where $Child\_D(j)$ is the set of membrane j children that can be dissolved.

Normalized total context for membrane j. This is defined as the $TC(j)$ sorted in depth and in pre-order:

$$TC_{Normal}(j) = (j_1, TC_{Normal}(j_1), \ldots, j_n, TC_{Normal}(j_n)), \tag{2}$$

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 no dissolving capability. For instance, in our P system, $TC_{Normal}(3) = \{6, 8, 9, 7\}$.

Each usefulness state 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 for 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 membrane j. Considering formula 2, the usefulness state for membrane j can be expressed in a deeper way as $q^j(t) = (i_1, \ldots, i_k, TC_{Normal}(i_k), \ldots, i_n)$. Then, the transition obtained for membrane j is $q^j(t+1) = (i_1, \ldots, 0, q^{i_j}(t), \ldots, i_n)$

In the example, if membrane 3 is in 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 describe the succession 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 for every membrane of the system.
2. Active rules are applied in a maximally parallel and non-deterministic way in 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 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.

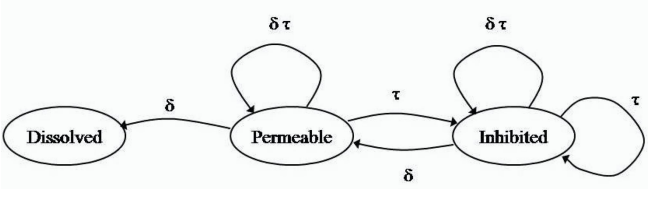


Fig. 3. Membrane permeability states

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:

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 to 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 Subsection 2.2.

In order to achieve these goals, the proxy must know the membrane structure, as regards membranes allocated in the proxy processor. For each 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:

$$TCL(j) = \sum_{k=1}^{n} (1 \; + \; TCL'(j_k)) \tag{3}$$

where $j_k \in Child\_Of(j)$ and

$$TCL'(j_k) = \begin{cases} TCL(j_k) \text{ if } j_k \text{ has dissolving capability} \\ 0 \qquad\qquad\quad \text{otherwise} \end{cases}$$

- **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:

$$PTCF(j_i) = 1 \; + \; \sum_{k=1}^{i-1} (1 \; + \; TCL'(j_k)), \tag{4}$$

  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 child 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 1, Figure 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.

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 non-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 changed its permeability state and this is done through a XOR between the usefulness state and the $USM(j)$ field. As shown in Subsection 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

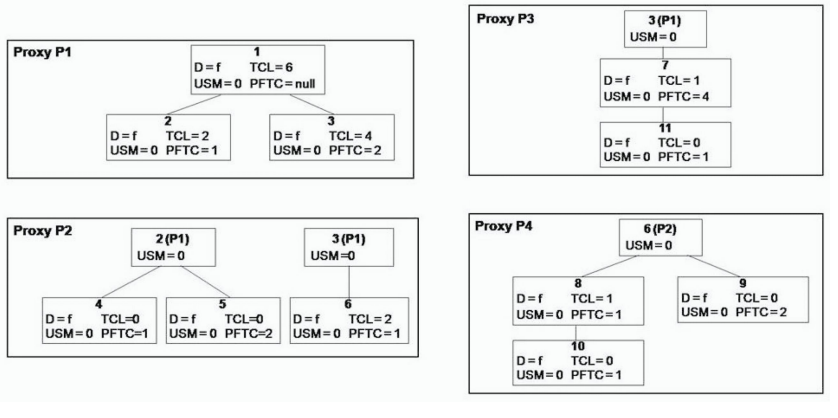


Fig. 4. Information stored in proxies to update usefulness states

that the 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 shows 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 2 shows, and necessarily these bits have to be replaced with the usefulness state of the dissolved membrane. 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 $TC_{Normal}(6) = \{8, 9\}$, 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 and membrane 6 was already dissolved in a previous step, in such a way that membrane 3 is the father of membrane 9. The $USM(3)$ field can be obtained in the following way:

Information = 1 (membrane 9, followed by its usefulness state)
Position = $PFTC(9) + PFTC(6) = 3$
Length = $TCL(3) = 4$

$USM(3) = 0010$ (as $TC_{Normal}(3) = \{6, 8, 9, 7\} \Rightarrow 9$ dissolution)

When a membrane j changes its permeability, the algorithm *ChangeUS* (Figure 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 child membranes change their permeability, involving the same father. Therefore, the usefulness state of

```
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. 5. Algorithm used to obtain the USM field for membrane j father

a membrane has to be modified with several masks. No matter the order in which the proxy processes permeability changes, commutative 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 before, the resulting $USM(3)$ is processed as follows:

$$\begin{cases} 1^{st}\ \text{ChangeUS(6, dissolution, 11)} \left\{ \begin{array}{l} \text{Father = 3} \\ \text{Mask = 1110} \end{array} \right\} \text{USM(3) = 1110} \\ 2^{nd}\ \text{ChangeUS(9, dissolution, -)} \left\{ \begin{array}{l} \text{Father = 3} \\ \text{Mask = 0010} \end{array} \right\} \text{USM(3) = 1110 XOR 0010 = 1100} \end{cases}$$

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 processed as follows:

$$\begin{cases} 1^{st}\ \text{ChangeUS(9, dissolution, -)} \left\{ \begin{array}{l} \text{Father = 6} \\ \text{Mask = 01} \end{array} \right\} \text{USM(6) = 01} \\ 2^{nd}\ \text{ChangeUS(6, dissolution, 11)} \left\{ \begin{array}{l} \text{Father = 3} \\ \text{Mask = (111 XOR (0+01)) + 0 = 1100} \\ \qquad\qquad\qquad\quad \uparrow \\ \qquad\qquad\qquad\quad \text{USM(6)} \end{array} \right\} \text{USM(3) = 1100} \end{cases}$$

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 in Figure 5 deals with this situation in lines 8 and 13, in which *OutProcessor* checks if the target membrane is allocated in other processor.

4 Encoding Targets of 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; u is called the *antecedent* and $v, v\delta, v\tau$ the *consequent* of rules.

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 space necessary 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 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.

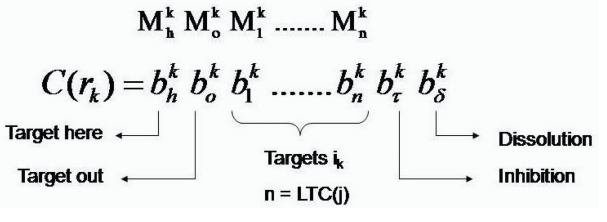


Fig. 6. Encoding a rule consequent

Table 2. Encoding consequent of membrane 3 rules

Rule	**Encoding**	**Multisets**
$r_1: a^2b \rightarrow (a^2\ in_7)(b\ in_6)\ \tau$	00100110	$M_1^1 = b,\ M_4^1 = a^2$
$r_2: a^2b^2 \rightarrow (a^2\ in_7)(a^2\ here)(b^2\ here)$	10000100	$M_h^2 = a^2b^2,\ M_4^2 = a^2$
$r_3: a^2b^5 \rightarrow (b^2\ in_6)(b^2\ out)\ \delta$	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 2 contains the encoded consequent of rules in membrane 3 of our example. Let us remind that the normalized total context for this membrane is {6,8,9,7}

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 consequent $(b_h\ b_o\ b_1\ \ldots\ b_n\ b_\tau\ b_\delta)$

$$C(p) \ = \ OR_{\forall\, r_i \,\in\, M_R(p)}\ C(r_i) \tag{5}$$

– $M_h(p)$, $M_o(p)$, $M_1(p)$, …, $M_n(p)$, the list of multisets of objects associated to $C(p)$.

$$M_h(p) \ = \sum_{\forall\, r_i \,\in\, M_R(p)} n_i\, M_h^i \tag{6}$$

$$M_o(p) \ = \sum_{\forall\, r_i \,\in\, M_R(p)} n_i\, M_o^i \tag{7}$$

$$M_j(p) \ = \sum_{\forall\, r_i \,\in\, M_R(p)} n_i\, M_j^i \tag{8}$$

– And finally, $M_a(p)$ the antecedent of the multiset of evolution rules.

$$M_a(p) \ = \sum_{\forall\, r_i \,\in\, M_R(p)} n_i\, M_a^i \tag{9}$$

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:

$$w \ = \ w \ - \ M_a(p) \ + \ M_h(p) \tag{10}$$

- $b_o\, b_1\, \ldots\, b_n$, 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 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 looking for the first non-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 analyze the normalized total context of the source membrane. Considering equation (2) for the 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 a consequence, in order to perform targeting search, the proxy has to store the following information related to membranes allocated to 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 in different processors. Hence, the proxy has to prepare properly some information to send them, because search of targets must continue on these processors. So, the proxy has to store some data about membranes placed in other processors with which there are established connection (virtual connections in Figure 1). The needed data for the proxy are:

- **DC(j)**, **TCL(j)**, and **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 7 shows the required information by proxies of the processors depicted in Figure 2.

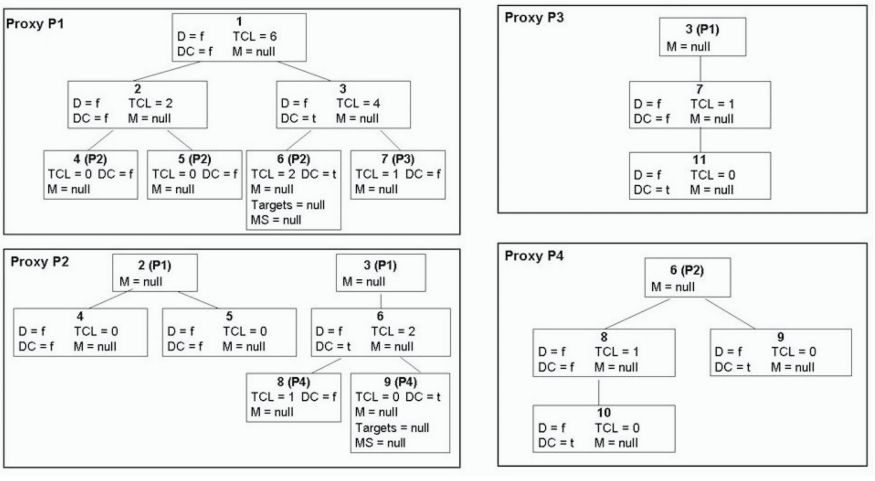


Fig. 7. Information stored in proxies to search targets

5.1 Target Search for $M_o(p)$

The algorithm presented here (*Target_Out*) looks for membrane j father 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 considers the situation in which the search has to be continued in another processor, then the partial result remains in a field M awaiting to be sent to the appropriate 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. 8. Target search for multiset $M_o(p)$

5.2 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 no dissolving capability ($DC(j)$) the analysis of this branch of the sub-tree is finished.

The recursive algorithm in Figure 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 child 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 that the child membrane were allocated to 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 that if $b_i = 1$ and the child membrane has dissolving capability, the algorithm skips the total context of the current child membrane, because these membranes are not possible targets (line 8).

```
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. 9. Searching targets for multisets $M_1(p)$ to $M_n(p)$

5.3 Membrane Dissolution

As it has been said 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:

1. Self processing in membrane: Objects remaining in the membrane after 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 behavior is reached by adding lines 5 to 9 to the *ChangeUS* algorithm, as Figure 10 shows.

```
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. 10. In case of dissolution, $M(j)$ is sent to membrane j father

(b) In case of objects for membrane j arriving in the 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 9).

6 Results Distribution

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 to other processor. In order to deliver these results, the distributed architecture in which the P system is implemented is very important, because external communications are implemented by distributed architectures in different ways.

6.1 Architecture Proposed by Tejedor et al. in [8]

The external communications are established in depth in the processors tree. After receiving information coming from the upper level, a processor P communicates with each descendant processor in both directions and from left to right; 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:

$$w \leftarrow w + M(j)$$

$$Usefulness\ State \leftarrow Usefulness\ State\ \ XOR\ \ USM(j).$$

6.2 Architectures Proposed by Bravo et al. in [2] and [3]

These architectures make use of one [2] or several [3] master processors which are in charge of controlling communications among slaves processors, while membranes 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 incoming 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, only usefulness states changes in father membranes in order to reconfigure the membrane structure of P systems are necessary.

The work presented here shows that usefulness states can be implemented in several distributed architectures for P systems implementations [8], [2] and [3]. In addition, usefulness states solve permeability changes in P 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 P 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 for the 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].

References

1. Arroyo, F., Luengo, C., Castellanos, J., de Mingo, L.F.: A binary data structure for membrane processors: Connectivity arrays. In: Alhazov, A., Martín-Vide, C., Păun, G. (eds.) Pre-proceedings of the Workshop on Membrane Computing, Tarragona, Spain, pp. 41–52 (2003)
2. Bravo, G., Fernández, L., Arroyo, F., Tejedor, J.: Master-slave distributed architecture for membrane systems implementation. In: 8th WSEAS Int. Conf. on Evolutionary Computing, EC 2007, Vancouver, Canada (June 2007)
3. Bravo, G., Fernández, L., Arroyo, F., Peña, M.A.: Hierarchical master-slave architecture for membrane systems implementation. In: 13th Int. Symposium on Artificial Life and Robotics, AROB 2008, Beppu, Japan (February 2008)
4. Ciobanu, G., Wenyuan, G.: P systems running on a cluster of computers. In: Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2003. LNCS, vol. 2933, pp. 123–139. Springer, Heidelberg (2004)
5. Frutos, J.A., Fernández, L., Arroyo, F., Bravo, G.: Static analysis of usefulness states in transition P systems. In: Proceedings of the Fifth International Conference, Information Research and Applications, I.TECH 2007, Varna, Bulgary, pp. 174–182 (June 2007)
6. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61, 108–143 (2000)
7. Syropoulos, A., Mamatas, E.G., Alliomes, P.C., et al.: A distributed simulation of P systems. In: Alhazov, A., Martín-Vide, C., Păun, G. (eds.) Preproceedings of the Workshop on Membrane Computing, Tarragona, Spain, pp. 455–460 (2003)
8. Tejedor, J., Fernández, L., Arroyo, F., Bravo, G.: An architecture for attacking the bottleneck communication in P systems. In: Sugisaka, M., Tanaka, H. (eds.) Proceedings of the 12th Int. Symposium on Artificial Life and Robotics, Beppu, Japan, pp. 500–505 (January 2007)

A P-Lingua Programming Environment for Membrane Computing

Daniel Díaz-Pernil, Ignacio Pérez-Hurtado,
Mario J. Pérez-Jiménez, and Agustín Riscos-Núñez

Research Group on Natural Computing
Dpt. Computer Science and Artificial Intelligence, University of Sevilla
Avda. Reina Mercedes s/n. 41012 Sevilla, Spain
{sbdani,perezh,marper,ariscosn}@us.es

Abstract. 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, while 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 an 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 non-deterministic 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.

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 187–203, 2009.

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 inter-operability) 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 inter-operability. 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 self-containment 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 one of these models, *P systems with active membranes*. Such a system is a construct of the form $\Pi = (O, H, \mu, w_1, \ldots, w_m, R)$, 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 labeled with elements of H, $w_1, \ldots, w_m$ 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 labeled 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 labeled by $1, \ldots, p$, and initial multisets $w_1, \ldots, w_p$ associated with them; (b) Σ is an (input) alphabet strictly contained in Γ; the initial multisets are over $\Gamma - \Sigma$; 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 $(\mu, w_1, \ldots, w_{i_{\Pi}} + m, \ldots, w_p)$.

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 labeled 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 $[a \rightarrow v]_h^\alpha$ is given next:
   α`[a --> v]'h`
2. The format to define *send-in communication rules* of type $a\,[\,]_h^\alpha \rightarrow [b]_h^\beta$ is given next:
   `a`α`[]'h -->`β`[b]`
3. The format to define *send-out communication rules* of type $[a]_h^\alpha \rightarrow b[\,]_h^\beta$ 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 R contains the following rules:

$\{[d_k]_2^0 \rightarrow [d_k]_2^+[d_k]_2^- : 1 \leq k \leq n\}$

$\{[x_{i,1} \rightarrow r_{i,1}]_2^+, [\bar{x}_{i,1} \rightarrow r_{i,1}]_2^- : 1 \leq i \leq m\}$

$\{[x_{i,1} \rightarrow \lambda]_2^-, [\bar{x}_{i,1} \rightarrow \lambda]_2^+ : 1 \leq i \leq m\}$

$\{[x_{i,j} \rightarrow x_{i,j-1}]_2^+, [x_{i,j} \rightarrow x_{i,j-1}]_2^- : 1 \leq i \leq m, 2 \leq j \leq n\}$

$\{[\bar{x}_{i,j} \rightarrow \bar{x}_{i,j-1}]_2^+, [\bar{x}_{i,j} \rightarrow \bar{x}_{i,j-1}]_2^- : 1 \leq i \leq m, 2 \leq j \leq n\}$

$\{[d_k]_2^+ \rightarrow []_2^0 d_k, [d_k]_2^- \rightarrow []_2^0 d_k : 1 \leq k \leq n\}$

$\{d_k[]_2^0 \rightarrow [d_{k+1}]_2^0 : 1 \leq k \leq n-1\}$

$\{[r_{i,k} \rightarrow r_{i,k+1}]_2^0 : 1 \leq i \leq m, 1 \leq k \leq 2n-1\}$

$\{[d_k \rightarrow d_{k+1}]_1^0 : n \leq k \leq 3n-3\}$

$[d_{3n-2} \rightarrow d_{3n-1}e]_1^0,\ e[]_2^0 \rightarrow [c_1]_2^+,\ [d_{3n-1} \rightarrow d_{3n}]_1^0$

$\{[d_k \rightarrow d_{k+1}]_1^0 : 3n \leq k \leq 3n+2m+2\}$

$\{[r_{i,2n} \rightarrow r_{i-1,2n}]_2^- : 1 \leq i \leq m\}$

$[r_{1,2n}]_2^+ \rightarrow []_2^- r_{1,2n},\ r_{1,2n}[]_2^- \rightarrow [r_{0,2n}]_2^+$

$\{[c_k \rightarrow c_{k+1}]_2^- : 1 \leq k \leq m\}$

$[c_{m+1}]_2^+ \rightarrow []_2^+ c_{m+1}\ ,\ [c_{m+1} \rightarrow c_{m+2}t]_1^0,$

$[t]_1^0 \rightarrow []_1^+ t\ ,\ [c_{m+2}]_1^+ \rightarrow []_1^- Yes\ ,\ [d_{3n+2m+3}]_1^0 \rightarrow []_1^+ No$

4.1 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 process 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.

7.1 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 4.1. 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 labeled 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 software 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.

Acknowledgment

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.

References

1. Díaz–Pernil, D., Pérez–Hurtado, I., Pérez–Jiménez, M.J., Riscos–Núñez, A.: P-lingua: A programming language for membrane computing. In: Díaz–Pernil, D., et al. (eds.) Proc. 6th Brainstorming Week on Membrane Computing, Fénix Editora, pp. 135–155 (2008)
2. Freund, R., Verlan, S.: A formal framework for static (tissue) P systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 271–284. Springer, Heidelberg (2007)
3. Gutiérrez–Naranjo, M.A., Pérez–Jiménez, M.J., Riscos–Núñez, A.: Available membrane computing software. In: Ciobanu, G., Păun, G., Pérez-Jiménez, M.J. (eds.) Applications of Membrane Computing, pp. 411–436. Springer, Heidelberg (2006)
4. Gutiérrez–Naranjo, M.A., Pérez–Jiménez, M.J., Riscos–Núñez, A.: Towards a programming language in cellular computing. Electronic Notes in Theoretical Computer Science 123, 93–110 (2005)
5. Păun, G.: Computing with membranes. J. Computer and System Sci. 61, 108–143 (2000)
6. Pérez-Jiménez, M.J., Romero–Jiménez, A., Sancho–Caparrini, F.: Complexity classes in cellular computing with membranes. Natural Computing 2, 265–285 (2003)
7. Pérez–Jiménez, M.J., Romero–Jiménez, A., Sancho–Caparrini, F.: A polynomial complexity class in P systems using membrane division. J. Automata, Languages and Combinatorics 11, 423–434 (2006)
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] and Florentin Ipate[2]

[1] Department of Computer Science, The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
M.Gheorghe@dcs.shef.ac.uk
[2] Department of Computer Science, Faculty of Mathematics and Computer Science
The University of Piteşti
Str Targu din Vale 1, 110040 Piteşti
florentin.ipate@ifsoft.ro

Abstract. 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 [17]. 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 compartmentalization 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 organization and behavior 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)

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 204–216, 2009.

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, 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 correctness of this process and in mapping it back to P systems.

The paper is organized 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 and 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*, and* $h : Q \times A \longrightarrow Q$ *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 analyzed 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 behavior 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 utilized 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 compartmentalization of the entire model, parallel behavior, 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:

$$\Pi_i = (V_i, \mu_i, w_i, R_i), \text{ where:}$$

- $V_1 = V_2 = V_3 = V_4 = \{s, a, b, c\}$;
- $\mu_1 = \mu_2 = \mu_3 = \mu_4 = [_1\ ]_1$;
- $w_1 = w_2 = w_3 = w_4 = s$;
- $R_1 = \{r_1 : s \to ab,\ r_2 : a \to c,\ r_3 : b \to bc,\ r_4 : b \to c\}$;
- $R_2 = \{r_1 : s \to ab,\ r_2 : a \to \lambda,\ r_3 : b \to c\}$;
- $R_3 = \{r_1 : s \to ab,\ r_2 : a \to bcc,\ r_3 : b \to \lambda\}$;
- $R_4 = \{r_1 : s \to ab,\ r_2 : a \to bc,\ r_3 : a \to c,\ r_4 : b \to c\}$.

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:

- $S(\Pi_1) = \{(1,0,0,0), (0,1,1,0)\} \cup \{(0,0,k,n) \mid k = 0,1; n \geq 2\}$;
- $S(\Pi_2) = \{(1,0,0,0), (0,1,1,0), (0,0,0,1)\}$;
- $S(\Pi_3) = \{(1,0,0,0), (0,1,1,0), (0,0,1,2), (0,0,0,2)\}$;
- $S(\Pi_4) = \{(1,0,0,0), (0,1,1,0), (0,0,1,2), (0,0,0,2), (0,0,0,3)\}$.

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:

- $L(\Pi_1) = \{(0,0,0,n) \mid n \geq 2\}$;
- $L(\Pi_2) = \{(0,0,0,1)\}$;
- $L(\Pi_3) = \{(0,0,0,2)\}$;
- $L(\Pi_4) = \{(0,0,0,2),(0,0,0,3)\}$.

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 generalization 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 $w \Longrightarrow^* u_1 a v_1 \Longrightarrow u_1 ubvv_1 \Longrightarrow u_1 uyvv_1 \Longrightarrow^* z$; $z, u_1, v_1, u, v, y \in V^*$, $a, b \in V$. 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:

$$\Pi'_i = (V_i, \mu_i, w_{1,i}, w_{2,i}, R_{1,i}, R_{2,i}), \text{ where:}$$

- $V_1 = V_2 = \{s, a, b, c\}$;
- $\mu_1 = \mu_2 = [_1[_2\ ]_2]_1$;
- $w_{1,1} = s,\ w_{2,1} = \lambda,\ w_{1,2} = s,\ w_{2,2} = \lambda$;
- $R_{1,1} = \{r_1 : s \to sa(b, in),\ r_2 : s \to ab,\ r_3 : b \to a,\ r_4 : a \to c\}$;
- $R_{2,1} = \{r_1 : b \to bc,\ r_2 : b \to c\}$;
- $R_{1,2} = \{r_1 : s \to sa(b, in),\ r_2 : s \to ab(b, in)(c, in),\ r_3 : b \to a,\ r_4 : a \to c\}$;
- $R_{2,2} = \{r_1 : b \to \lambda,\ r_2 : b \to c\}$.

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 $S_1 = \{r_1 : s \to sa(b, in),\ r_2 : s \to ab,\ r_3 : b \to a\}$ and $S_2 = \emptyset$;
- $Occs_2(\Pi'_1, b) = S_1 \cup S_2$, where $S_1 = \{r_1 : s \to sa(b, in)\}$ and $S_2 = \{r_1 : b \to bc\}$.

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 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.

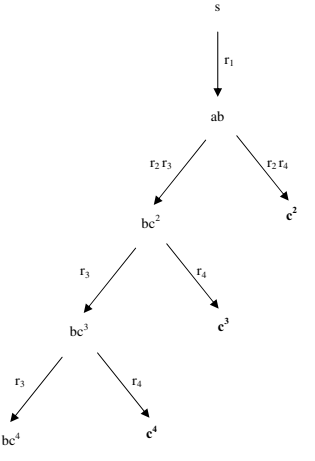


Fig. 1. Derivation tree for Π_1 and $k = 4$

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 in [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, the minimization of DFCA can be approached as an inference problem ([8]), 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 2; q_3 in Figure 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.

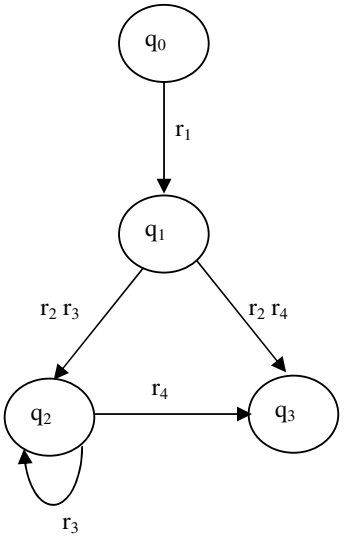


Fig. 2. Minimal DFCA for Π_1 and $k = 4$

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 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 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 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.

For $k = 3$, the derivation tree of Π_1' defined above is as represented in Figure 3. For clarity, in Figure 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.

Similar to one compartment case, test sets for considered criteria, state and transition cover, can be defined in this more general context.

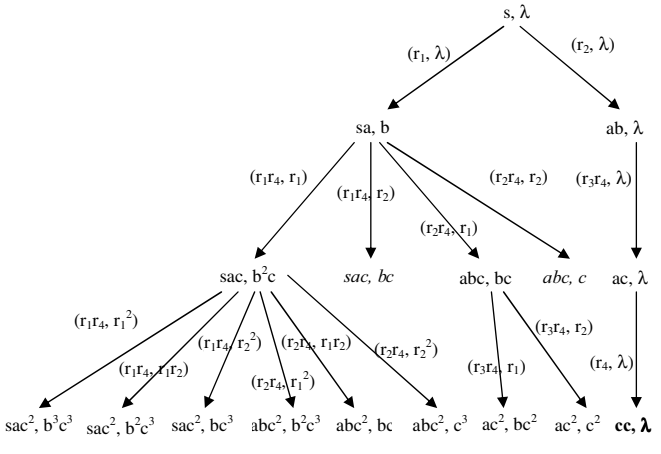


Fig. 3. Derivation tree for Π_1' and $k = 3$

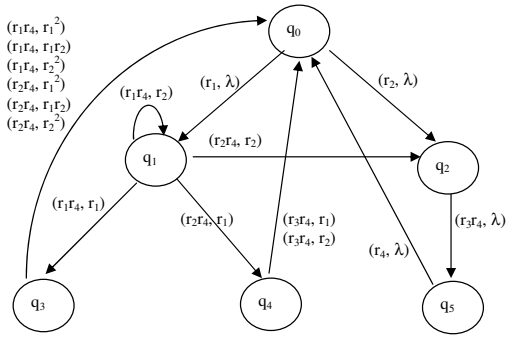


Fig. 4. Minimal DFCA for Π_1' and $k = 3$

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.

References

1. Andrei, O., Ciobanu, G., Lucanu, D.: Structural operational semantics of P systems. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 31–48. Springer, Heidelberg (2006)
2. Andrei, O., Ciobanu, G., Lucanu, D.: A rewriting logic framework for operational semantics of membrane systems. Theoretical Computer Sci. 373, 163–181 (2007)

3. Câmpeanu, C., Păun, A., Smith, J.R.: Incremental construction of minimal deterministic finite cover automata. Theoretical Computer Sci. 363, 135–148 (2006)
4. Câmpeanu, C., Păun, A., Yu, S.: An efficient algorithm for constructing minimal cover automata for finite languages. Intern. J. Foundation of Computer Sci. 13, 83–97 (2002)
5. Câmpeanu, C., Sântean, N., Yu, S.: Minimal cover-automata for finite languages. In: Champarnaud, J.-M., Maurel, D., Ziadi, D. (eds.) WIA 1998. LNCS, vol. 1660, pp. 43–56. Springer, Heidelberg (1999)
6. Câmpeanu, C., Sântean, N., Yu, S.: Minimal cover-automata for finite languages. Theoretical Computer Sci. 267, 3–16 (2002)
7. Ciobanu, G., Păun, G., Pérez-Jiménez, M.J. (eds.): Applications of Membrane Computing. Springer, Heidelberg (2006)
8. García, P., Ruiz, J.: A note on the minimal cover-automata for finite languages. Bulletin of the EATCS 83, 193–194 (2004)
9. Gold, M.E.: Complexity of automaton identification from given data. Information and Control 37, 302–320 (1978)
10. Holcombe, M., Ipate, F.: Correct Systems – Building Business Process Solutions. Springer, Heidelberg (1998)
11. Körner, H.: On minimizing cover automata for finite languages in O(n log n) time. In: Champarnaud, J.-M., Maurel, D. (eds.) CIAA 2002. LNCS, vol. 2608, pp. 117–127. Springer, Heidelberg (2003)
12. Körner, H.: A time and space efficient algorithm for minimizing cover automata for finite languages. Intern. J. Foundation of Computer Sci. 14, 1071–1086 (2003)
13. Lämmel, R.: Grammar testing. In: Hussmann, H. (ed.) FASE 2001. LNCS, vol. 2029, pp. 201–216. Springer, Heidelberg (2001)
14. Li, H., Jin, M., Liu, C., Gao, Z.: Test criteria for context-free grammars. COMPSAC 1, 300–305 (2004)
15. Oncina, J., García, P.: Inferring regular languages in polynomial update time. In: de la Blanca, N.P., Sanfeliu, A., Vidal, E. (eds.) Pattern Recognition and Image Qnalysis, pp. 49–61. World Scientific, Singapore (1992)
16. Păun, A., Sântean, N., Yu, S.: An $O(n^2)$ algorithm for constructing minimal cover automata for finite languages. In: Yu, S., Păun, A. (eds.) CIAA 2000. LNCS, vol. 2088, pp. 243–251. Springer, Heidelberg (2001)
17. Păun, G.: Computing with membranes. J. Computer and System Sci. 61, 108–143 (2000)
18. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
19. The P Systems Web Site, `http://ppage.psystems.eu`

Hebbian Learning from Spiking Neural P Systems View

Miguel A. Gutiérrez-Naranjo and Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
{magutier,marper}@us.es

Abstract. *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) [12]*

Neuroscience has been a fruitful research area since the pioneering work of Ramón y Cajal in 1909 [20] 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 [8] and also in artificial neural networks [11], it is usual to consider a parameter 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 and 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

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 217–230, 2009.

signal or whether the code can be read by external observers who can understand the message. At present, a definite answer to these questions is not known.

Since all spikes (short electrical pulses) 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., [17] or [13]) 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 [21]. 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, [9] or [5]). 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 [14] with the aim of incorporating in membrane computing (more information can be found at [22]) 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 by the action potentials, which are encoded by objects of the same 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.

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 a corresponding software is shown in Section 4. Finally, some conclusions 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 [19] 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 enough spikes 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.

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 $\boldsymbol{x} = (x_1, \ldots, x_m)$ 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 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

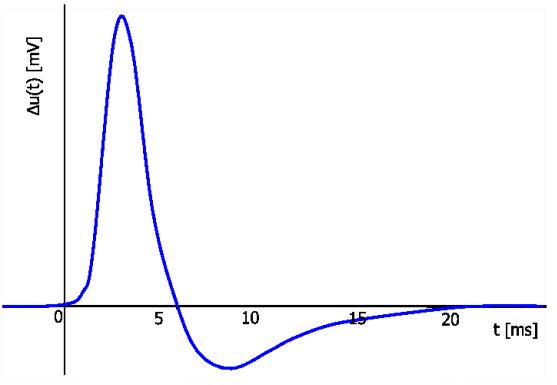


Fig. 1. Dynamics of one spike

enough to activate the postsynaptic neuron, it decays after some milliseconds and the neuron comes back to its resting potential (see Fig. 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 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 shows the changes of potential in the postsynaptic neuron until 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 can be seen as a generalization of the decaying spikes presented in [6], 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 to a neuron, e is decremented

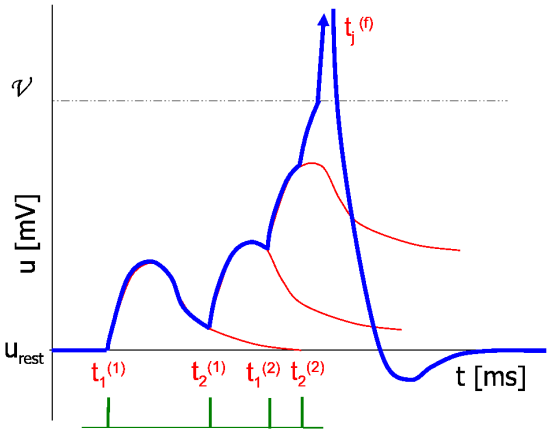


Fig. 2. The potential at the postsynaptic neuron

by one in each step of computation. As soon as $e = 0$, the corresponding spike is lost and cannot be used anymore.

In this way, a rule $E/a^k \to a^p; d$ $(k > p)$ where a^p are p decaying spikes (a, e) can be seen with our notation as $E/a^k \to (a^p, S); d$ 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* $a^k \to (a^{n_{ij}}, S); d_{ij}$

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 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 only contains*[1] *the rule* $a^p a^*/a^p \rightarrow a; 0$. *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* $\boldsymbol{x} = (x_1, \ldots, x_m)$ *of* m *non-negative integers* x_i.

A Hebbian SN P system unit with input *is a pair* $(H\Pi, \boldsymbol{x})$ *where* $H\Pi$ *is Hebbian SN P system unit and* $\boldsymbol{x}$ *is an input for it.*

The intuitive idea behind the input is encoding the information in time. Each component of the input represents the moment, according to the global clock, in which k spikes are provided to the corresponding presynaptic neuron.

2.4 How It Works

Next 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 $\boldsymbol{x} = (x_1, \ldots, x_m)$ represents the time in which k spikes are provided to the neuron 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 minimum 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

[1] This rule is an adaptation of the concept of a rule from an extended spiking neural P system with thresholds taken from [6].

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 $t = x_i + d_{ij} + 1 + h$ is s_{k+h}, if $h \in \{0, \ldots, r-l\}$ 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 $\boldsymbol{x}$ and on the rules chosen in each neuron $R^*(R_{1i_1}, \ldots, R_{mi_m}, \boldsymbol{x}, t) \in \mathbb{N}$. 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, $a^p a^* / a^p \rightarrow a$, 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 $\boldsymbol{x}$ produces or not an output[2].

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 [7], 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

[2] A detailed description and some examples can be found in [10].

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 [3] and [12]). The weights of synapses such that their contributions arrive to the postsynaptic neuron *after* the postsynaptic spike is expelled are decreased (see [4] and [15]). This is basically the learning mechanism suggested in [16].

3.1 The Model

In order to implement a learning algorithm in our Hebbian SN P system unit, 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 theory of artificial neural networks [11]. 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 [18] 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 [7]. 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 $(EH\Pi, \boldsymbol{x})$, where $H\Pi$ is an Hebbian SN P system unit and $\boldsymbol{x}$ is an input for it.*

The semantics. As we saw before, each x_i in the input $\boldsymbol{x} = (x_1, \ldots, x_m)$ 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 generalization 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 $a^k \to (a^{n_{ij}}, S); d_{ij}$ 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 $a^k \to (a^{n_{ij}}, S); d_{ij}$ is activated and the efficacy of its synapse is represented by the weight w_{ij}. When the rule $a^k \to (a^{n_{ij}}, S); d_{ij}$ 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 time $t_0 + d + 1$, 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, $a^p a^* / a^p \to a$, is triggered.

3.2 The Learning Problem

Let us come back to the Hebbian SN P system units. In such units, provided an input $\boldsymbol{x}$, 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.
- $X = \{\boldsymbol{x_1}, \ldots \boldsymbol{x_n}\}$ is a finite set of *inputs* of $EH\Pi$.

- $L : \mathbb{Z} \to \mathbb{Z}$ 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 $L : \mathbb{Z} \to \mathbb{Z}$ and a rate of learning ϵ. Let us consider an input $\boldsymbol{x}$ and we will denote by $t_{\boldsymbol{x}}$ the moment when the postsynaptic neuron reaches the potential for the trigger of the postsynaptic neuron. If such potential is not reached then $t_{\boldsymbol{x}} = \infty$.

On the other hand, for each rule $R_{ij} \equiv a^k \to (a^{n_{ij}}, S); d_{ij}$ of a presynaptic neuron we can compute the moment $t_{ij}^{\boldsymbol{x}}$ in which its contribution to the postsynaptic potential arrives to the postsynaptic neuron. It depends on the input $\boldsymbol{x}$ and the delay d_{ij} of the rule $t_{ij}^{\boldsymbol{x}} = \boldsymbol{x}_i + d_{ij} + 1$ where $\boldsymbol{x}_i$ is the i-th component of $\boldsymbol{x}$. 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 $t_{ij}^{\boldsymbol{x}}$ and the moment $t_{\boldsymbol{x}}$ in which the postsynaptic neuron is activated.

For each rule R_{ij} and each input $\boldsymbol{x}$, such a difference is $s_{ij}^{\boldsymbol{x}} = t_{\boldsymbol{x}} - t_{ij}^{\boldsymbol{x}}$

- If $s_{ij}^{\boldsymbol{x}} = 0$, 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 $s_{ij}^{\boldsymbol{x}} > 0$ 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 $s_{ij}^{\boldsymbol{x}} < 0$ or $s_{ij}^{\boldsymbol{x}} > 0$ 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 $\boldsymbol{x}$, $L(s_{ij}^{\boldsymbol{x}}) \in \mathbb{Z}$ measures de degree of influence of the contribution of R_{ij} to the activation of the postsynaptic neuron produced by the input $\boldsymbol{x}$.

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 $\boldsymbol{x}$ 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

$$w'_{ij} = w_{ij} + \epsilon \cdot L(s_{ij}^{\boldsymbol{x}})$$

Depending on the sign of $L(s_{ij}^{\boldsymbol{x}})$, the rule R_{ij} will increase or decrease its efficacy. Note that $L(s_{ij}^{\boldsymbol{x}})$ 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 $\boldsymbol{x} \in X$ and outputs $H\Pi'$. In the second step, the learning problem $(H\Pi', X - \{\boldsymbol{x}\}, L)$ 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 the rule $a^{1200}a^*/a^{1200} \to a; 0$.

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 $\epsilon = 0.001$

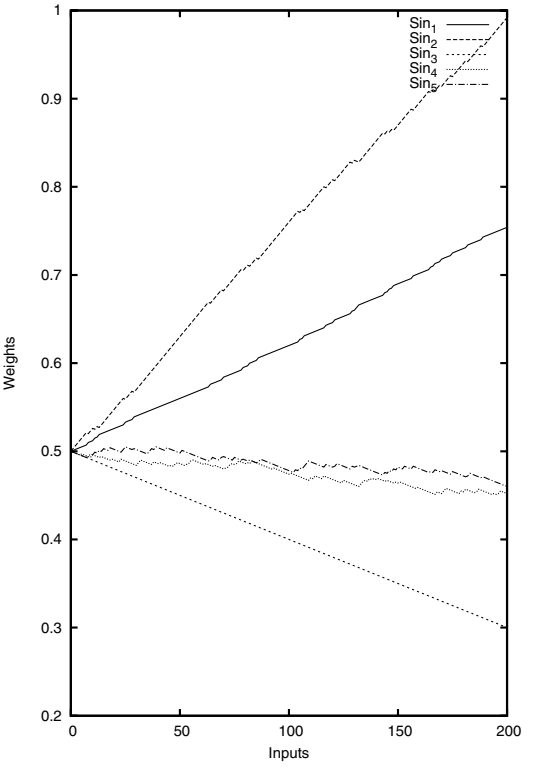


Fig. 3. The evolution of the weights

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 $w_{11} = 0.754$, $w_{12} = 0.992$, $w_{13} = 0.3$, $w_{21} = 0.454$, $w_{22} = 0.460$. Figure 3 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}.

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 has 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, a deeper study related to the input, the decay and the weights is needed.

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 a 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.

References

1. Adrian, E.D.: The impulses produced by sensory nerve endings: Part I. The Journal of Physiology 61, 49–72 (1926)
2. Adrian, E.D.: The Basis of Sensation. W.W. Norton, New York (1926)

3. Bliss, T.V.P., Collingridge, G.L.: A synaptic model of memory: long-term potentiation in the hippocampus. Nature 361, 31–39 (1993)
4. Debanne, D., Gähwiler, B.H., Thompson, S.M.: Asynchronous pre- and postsynaptic activity induces associative long-term depression in area CA1 of the rat hippocampus in vitro. Proc. National Academy of Sciences 91, 1148–1152 (1994)
5. Engel, A.K., König, P., Kreiter, A.K., Singer, W.: Interhemispheric synchronization of oscillatory neural responses in cat visual cortex. Science 252, 1177–1179 (1991)
6. Freund, R., Ionescu, M., Oswald, M.: Extended spiking neural P systems with decaying spikes and/or total spiking. In: Proc. Intern. Workshop Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, pp. 64–75 (2007)
7. Gerstner, W., Kempter, R., van Hemmen, L., Wagner, H.: A neuronal learning rule for sub-millisecond temporal coding. Nature 383, 76–78 (1996)
8. Gerstner, W., Kistler, W.: Spiking Neuron Models. Single Neurons, Populations, Plasticity. Cambridge University Press, Cambridge (2002)
9. Gray, C.M., König, P., Engel, A.K., Singer, W.: Oscillatory responses in cat visual cortex exhibit inter-columnar synchronization which reflects global stimulus properties. Nature 338, 334–337 (1989)
10. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: A first model for Hebbian learning with spiking neural P systems. In: Díaz-Pernil, D., et al. (eds.) Sixth Brainstorming Week on Membrane Computing, Fénix Editora, pp. 211–233 (2008)
11. Haykin, S.: Neural Networks. A Comprehensive Foundation. Macmillan College Publishin Company, Inc., Basingstoke (1994)
12. Hebb, D.O.: The Organization of Behavior. Wiley, New York (1949)
13. Hubel, D.H., Wiesel, T.N.: Receptive fields of single neurons in the cat's striate cortex. The Journal of Phisiology 148, 574–591 (1959)
14. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae 71, 279–308 (2006)
15. Markram, H., Sakmann, B.: 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)
16. Markram, H., Tsodyks, M.: Redistribution of synaptic efficacy between neocortical pyramidal neurons. Nature 382, 807–810 (1996)
17. Mountcastle, V.B.: Modality anf topographic properties of single neurons of cat's somatosensory cortex. Journal of Neurophysiology 20(1957), 408–434
18. Natschläger, T., Ruf, B.: Spatial and temporal pattern analysis via spiking neurons. Network: Computation in Neural Systems 9, 319–338 (1998)
19. Păun, G.: Twenty six research topics about spiking neural P systems. In: Gutiérrez-Naranjo, M.A., et al. (eds.) Fifth Brainstorming Week on Membrane Computing, Fénix Editora, Sevilla, pp. 263–280 (2007)
20. Ramón y Cajal, S.: Histologie du Systeme Nerveux de l'Homme et des Vertebre. A. Maloine, Paris (1909)
21. Thorpe, S., Fize, S., Marlot, C.: Speed of processing in the human visual system. Nature 381, 520–522 (1996)
22. P systems web page, `http://ppage.psystems.eu/`

Event-Driven Metamorphoses of P Systems

Thomas Hinze, Raffael Faßler, Thorsten Lenser,
Naoki Matsumaru, and 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

Abstract. 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 behavior. From a modeling 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 discretized 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], 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 characterized 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 modeling 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

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 231–245, 2009.

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 behavior.

Motivated by this question, we contribute to the specification of a framework able to incorporate correlated temporally "local" models whose dynamical behavior 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 *prioritization* of reaction rules and a principle of operation based on discretized *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: discretization of chemical kinetics and structural network dynamics. On the one hand, metabolic or cell signalling P systems like [14] describe the dynamical behavior of a fixed reaction network based on concentration gradients, numerically studied in [8]. Results of [10] present a discretization of Hill kinetics mainly employed for gene regulatory networks. Artificial chemistries were explored in [7] along with issues of computability [13] and prioritization 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 organized as follows: First we present a method for discretization 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. 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 behavior 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 behavior 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

$$\begin{aligned}
a_{1,1}\mathrm{S}_1 + a_{2,1}\mathrm{S}_2 + \ldots + a_{n,1}\mathrm{S}_n &\xrightarrow{\hat{k}_1} b_{1,1}\mathrm{S}_1 + b_{2,1}\mathrm{S}_2 + \ldots + b_{n,1}\mathrm{S}_n \\
a_{1,2}\mathrm{S}_1 + a_{2,2}\mathrm{S}_2 + \ldots + a_{n,2}\mathrm{S}_n &\xrightarrow{\hat{k}_2} b_{1,2}\mathrm{S}_1 + b_{2,2}\mathrm{S}_2 + \ldots + b_{n,2}\mathrm{S}_n \\
&\ldots \\
a_{1,h}\mathrm{S}_1 + a_{2,h}\mathrm{S}_2 + \ldots + a_{n,h}\mathrm{S}_n &\xrightarrow{\hat{k}_h} b_{1,h}\mathrm{S}_1 + b_{2,h}\mathrm{S}_2 + \ldots + b_{n,h}\mathrm{S}_n,
\end{aligned}$$

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.

Discretization: Corresponding P Systems Π_{PMA}

The general form of a P system Π_{PMA} emulating the dynamical behavior of chemical reaction systems with strict *prioritization* of reaction rules based on discretized *mass-action* kinetics is a construct $\Pi_{\mathrm{PMA}} = (V, \Sigma, [_1\,]_1, L_0, R)$, 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 component R. The finite set $R = \{r_1, \ldots, r_h\}$ subsumes the reaction rules while each reaction rule $r_i \in \langle E_i \rangle \times \langle P_i \rangle \times \mathbb{R}_+$ is composed of a finite multiset of reactants (educts) $E_i \subset V \times \mathbb{N}$, products $P_i \subset V \times \mathbb{N}$, and the kinetic constant $k_i \in \mathbb{R}_+$. Multiplicities of elements in E_i and R_i correspond with according stoichiometric factors.

Since we strive for a deterministic P system, a strict prioritization 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. Prioritization 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 behavior 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 $r_i = \Big(\{(e_1, a_1), \ldots, (e_\mu, a_\mu)\}, \{(q_1, b_1), \ldots, (q_v, b_v)\}, k_i\Big)$ with $\mathrm{supp}(E_i) = \{e_1, \ldots, e_\mu\}$ and $\mathrm{supp}(P_i) = \{q_1, \ldots, q_v\}$ 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 behavior 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:

$$
\begin{aligned}
L_{t,0} &= L_t \\
L_{t,i} &= L_{t,i-1} \ominus \Bigg\{ \Bigg(a, \Bigg\lfloor k_i \cdot |E_i \cap \{(a,\infty)\}| \cdot \\
&\qquad \prod_{b \in \mathrm{supp}(E_i)} |L_{t,i-1} \cap \{(b,\infty)\}|^{|E_i \cap \{(b,\infty)\}|} \Bigg\rfloor \Bigg) \;\Bigg|\; \forall a \in \mathrm{supp}(E_i) \\
&\qquad \wedge \left(|L_{t,i-1} \cap \{(c,\infty)\}| \geq |E_i \cap \{(c,\infty)\}| \ \forall c \in \mathrm{supp}(E_i) \right) \Bigg\} \\
&\uplus \Bigg\{ \Bigg(a, \Bigg\lfloor k_i \cdot |P_i \cap \{(a,\infty)\}| \cdot \\
&\qquad \prod_{b \in \mathrm{supp}(E_i)} |L_{t,i-1} \cap \{(b,\infty)\}|^{|E_i \cap \{(b,\infty)\}|} \Bigg\rfloor \Bigg) \;\Bigg|\; \forall a \in \mathrm{supp}(P_i) \\
&\qquad \wedge \left(|L_{t,i-1} \cap \{(c,\infty)\}| \geq |E_i \cap \{(c,\infty)\}| \ \forall c \in \mathrm{supp}(E_i) \right) \Bigg\} \\
L_{t+1} &= L_{t,h}.
\end{aligned}
$$

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 discretized 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 discretization, 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 discretization interval.

System Classification and Properties

Π_{PMA} belongs to deterministic P systems with symbol objects, strict prioritization 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 behavior.

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 operate below 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 increment, decrement, 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 together with kinetic parameters are replaced. Thirdly, the subsequent system is set into operation.

For formulation of the transition framework, we utilize *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:

$$
\begin{aligned}
Q &= \{\Pi_{\mathrm{PMA}}^{(j)} \mid (j \in A) \wedge (A \subseteq \mathbb{N})\} \\
T &\subseteq \{(t = \tau) \mid (\tau \in B) \wedge (B \subseteq \mathbb{N})\} \cup \\
&\quad \{([a]\, \mathit{CMP}\, \kappa) \mid (\mathit{CMP} \in \{<, \leq, =, \neq, \geq, >\}) \wedge (\kappa \in \mathbb{N}) \wedge (a \in V^{(j)}) \wedge \\
&\qquad (\Pi_{\mathrm{PMA}}^{(j)} = (V^{(j)}, \Sigma^{(j)}, [\,]_1, L_0^{(j)}, R^{(j)}) \in Q) \wedge (j \in A)\}.
\end{aligned}
$$

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 $\Pi_{\mathrm{PMA}}^{(j)} \overset{c}{\mapsto} \Pi_{\mathrm{PMA}}^{(m)} \in \Delta$ from $\Pi_{\mathrm{PMA}}^{(j)} = (V^{(j)}, \Sigma^{(j)}, [\,]_1, L_0^{(j)}, R^{(j)})$ to $\Pi_{\mathrm{PMA}}^{(m)} = (V^{(m)}, \Sigma^{(m)}, [\,]_1, L_0^{(m)}, R^{(m)})$ 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 $\Pi_{\mathrm{PMA}}^{(j)} \overset{c}{\mapsto} \Pi_{\mathrm{PMA}}^{(m)}$, new reactions might appear restricted to reactants and products available in $V^{(m)}$. New reactions $r_i \in \langle V^{(m)} \times \mathbb{N} \rangle \times \langle V^{(m)} \times \mathbb{N} \rangle \times \mathbb{R}_+$ with unique priority index i become accumulated by the multiset $\mathit{AdditionalReactions}_{(j,m)}$. Accordingly, we consider vanishing reactions present in multiset $\mathit{VanishedReactions}_{(j,m)}$. The scheme

$$
\begin{aligned}
V^{(m)} &= V^{(j)} \cup \mathit{AdditionalSpecies}V_{(j,m)} \setminus \mathit{VanishedSpecies}V_{(j,m)} \\
\Sigma^{(m)} &= \Sigma^{(j)} \cup \mathit{AdditionalSpecies}\Sigma_{(j,m)} \setminus \mathit{VanishedSpecies}\Sigma_{(j,m)}
\end{aligned}
$$

$$L_0^{(m)} = L_t^{(j)} \uplus \{(a,0) \mid a \in \mathit{AdditionalSpecies}V_{(j,m)}\}$$
$$R^{(m)} = R^{(j)} \uplus \mathit{AdditionalReactions}_{(j,m)} \ominus \mathit{VanishedReactions}_{(j,m)}$$

decomposes the P system transition into all single components. After performing the transition, the obtained system $\Pi_{\mathrm{PMA}}^{(m)}$ includes reactions $R^{(m)} = \{r_i \mid (i \in A) \wedge (A \subset \mathbb{N})\}$ where A is an arbitrary finite subset of natural numbers. In order to preserve the strict prioritization 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 modularization 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 $\#_j$ (non-negatively 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-initialization 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 $c = \mathit{const.}$ The boolean value of the variable x is determined whenever one of the following conditions is fulfilled: The inequality $[X^T] \ll [X^F]$ indicates "false" ($x = 0$) and $[X^F] \ll [X^T]$ "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.

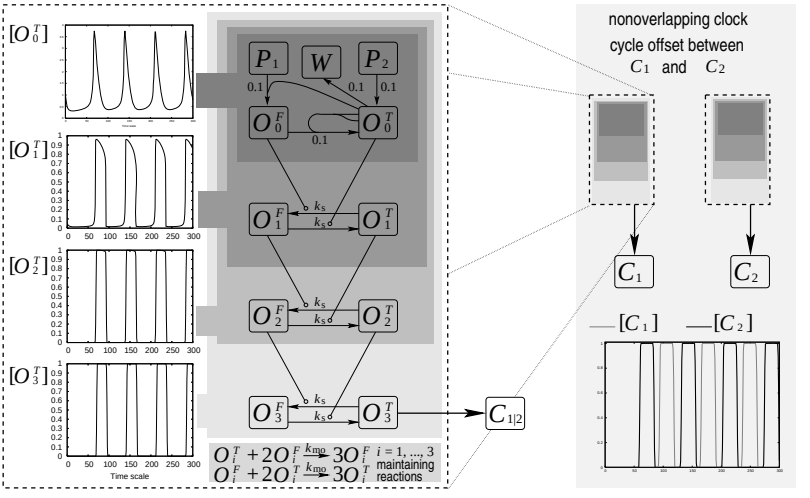


Fig. 1. Generation of chemical clock signals $[C_1]$, $[C_2]$ (right) by cascadization of toggle switches (left)

A Chemical Clock by Extending an Oscillating Reaction Network

A chemical implementation of a clock is necessary in order to synchronize 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 extended oscillating reaction network provides all clock signals. As preferred network template for permanent oscillation, we adopt the well-studied Belousov-Zhabotinsky reaction [2,21] depicted in the upper-left part of Figure 1 whose dynamical behavior 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 normalization 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 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 initializations (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 non-overlapping clock signals whose offset constitutes approximately one half of the clock cycle, see Figure 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

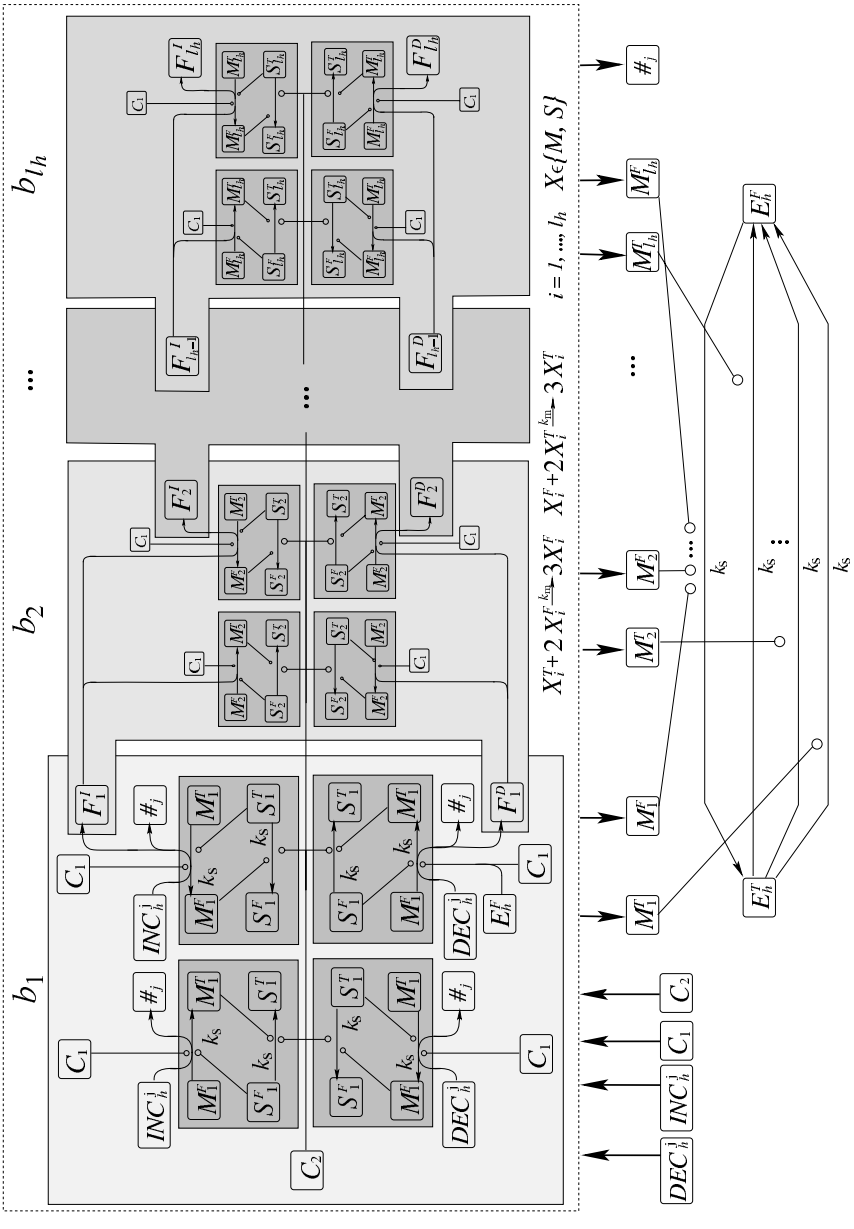


Fig. 2. Chemical reaction network of a register capable of processing a bitwise extendable binary number $\ldots b_{l_h} b_{l_h-1} \ldots b_2 b_1$ with $b_\alpha \in \{0, 1\}$ including interfaces for micro-operations increment, nonnegative decrement, and comparison to zero

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 finalizes 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 2) covers this task.

With regard to the functionality of a register machine, a sequence of interconnected MSFFs represents a register. Interconnections between neighbored MSFFs reflect the capability of incrementing and decrementing register contents. In case of increment, 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 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 2.

Decrement is organized 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 decrement of binary strings $0 \ldots 0$. Figure 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.

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* $\#_j$

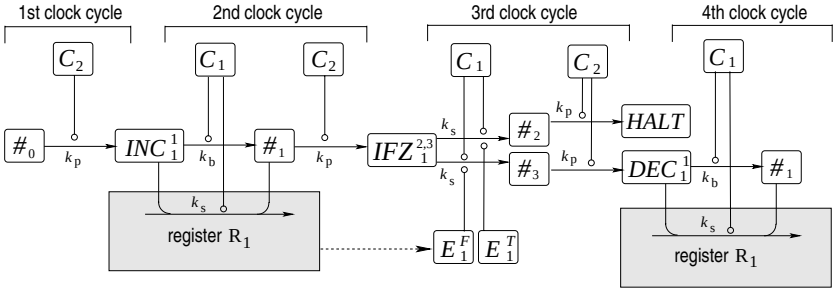


Fig. 3. Example: Chemical program control for $M = (\{\mathrm{R}_1\}, \{\#_0, \ldots, \#_3\}, P, \#_0)$ with $P = \{\#_0 : \mathrm{INC}\ \mathrm{R}_1\ \#_1, \quad \#_1 : \mathrm{IFZ}\ \mathrm{R}_1\ \#_2\ \#_3, \quad \#_2 : \mathrm{HALT}, \quad \#_3 : \mathrm{DEC}\ \mathrm{R}_1\ \#_1\}$

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 sequentialize 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 in Table 1.

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 2. Here,

Table 1.

instruction	reactions
$\#_i : \mathrm{INC}\ \mathrm{R}_h\ \#_j$	$\#_i + C_2 \xrightarrow{k_\mathrm{p}} INC_h^j + C_2$ $INC_h^j + C_1 \xrightarrow{k_\mathrm{b}} \#_j + C_1$
$\#_i : \mathrm{DEC}\ \mathrm{R}_h\ \#_j$	$\#_i + C_2 \xrightarrow{k_\mathrm{p}} DEC_h^j + C_2$ $DEC_h^j + C_1 \xrightarrow{k_\mathrm{b}} \#_j + C_1$
$\#_i : \mathrm{IFZ}\ \mathrm{R}_h\ \#_j\ \#_q$	$\#_i + C_2 \xrightarrow{k_\mathrm{p}} IFZ_h^{j,q} + C_2$ $IFZ_h^{j,q} + E_h^T + C_1 \xrightarrow{k_\mathrm{s}} \#_j + E_h^T + C_1$ $IFZ_h^{j,q} + E_h^F + C_1 \xrightarrow{k_\mathrm{s}} \#_q + E_h^F + C_1$
$\#_i : \mathrm{HALT}$	$\#_i + C_2 \xrightarrow{k_\mathrm{p}} HALT + C_2$

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 non-negatively. Instruction species of the form $IFZ_h^{j,q}$ utilize 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 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 initialization $(\mathrm{R}_1, \mathrm{R}_2) := (2, 1)$ on extendable bit word registers emulates a case study of the integer addition "$2 + 1$" whose dynamical behavior 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 (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 increment 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 (lower part). This system contains additional species and reactions (according to Figure 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.

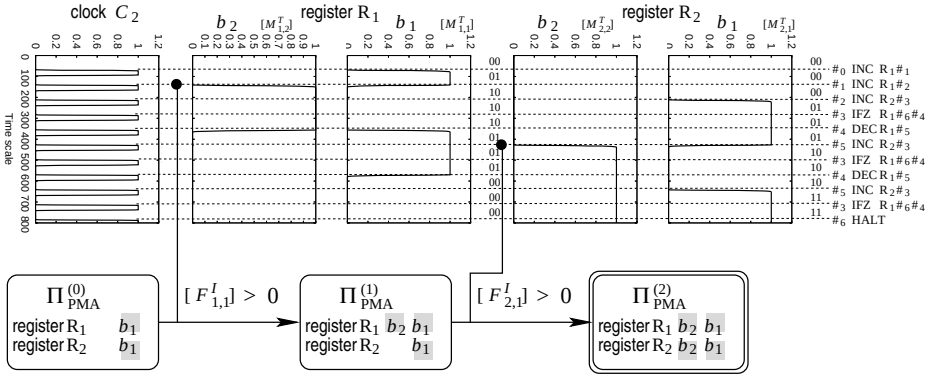


Fig. 4. Dynamical behavior of a chemical register machine acting as an adder (upper part) and transitions between corresponding P systems successive enlarging storage capacity (lower part)

All simulations of the dynamical register machine behavior 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 behavior is a powerful tool to automatically devise complex systems capable of computational tasks. We have designed and implemented a software (SBMLevolver) [12] for evolutionary 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 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}.

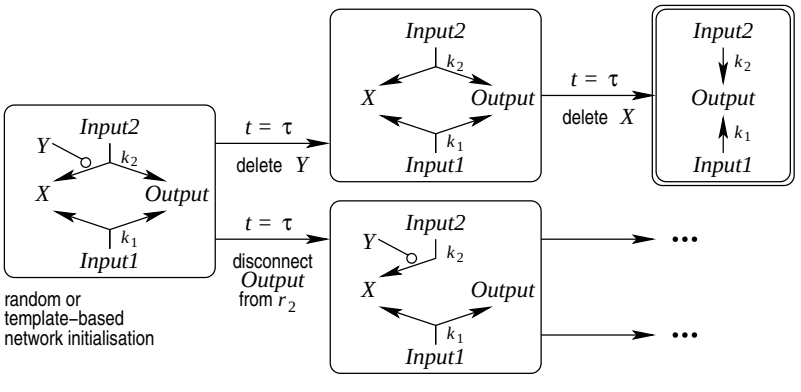


Fig. 5. Part of a state transition system sketching the trace of an artificial structural evolution towards a reaction network for addition of two numbers given as initial concentrations of species *Input1* and *Input2*. In the SBMLevolver, each network passes a separate supplementary parameter fitting (optimization) of kinetic constants (not shown).

6 Conclusions

The formalization 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 modeling 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.

References

1. Alon, U.: An Introduction to Systems Biology. Chapman & Hall, Boca Raton (2006)
2. Belousov, B.P.: A periodic reaction and its mechanism. Compilation of Abstracts in Radiation Medicine 147, 145 (1959)
3. Blankenship, R.E.: Molecular Mechanisms of Photosynthesis. Blackwell Science, Malden (2002)
4. Brody, H.M., et al.: Phenotypic Plasticity. Oxford University Press, Oxford (2003)
5. Connors, K.A.: Chemical Kinetics. VCH Publishers, Weinheim (1990)
6. Debanne, D.: Brain plasticity and ion channels. Journal of Physiology 97, 403–414 (2003)
7. Dittrich, P., et al.: Artificial chemistries. A review. Artificial Life 7, 225–275 (2001)
8. Fontana, F., et al.: Discrete solutions to differential equations by metabolic P systems. Theor. Comput. Sci. 372, 165–182 (2007)
9. Funahashi, A., et al.: CellDesigner: a process diagram editor for gene-regulatory and biochemical networks. Biosilico 1, 159–162 (2003), `www.celldesigner.org`
10. Hinze, T., Hayat, S., Lenser, T., Matsumaru, N., Dittrich, P.: Hill kinetics meets P systems: A case study on gene regulatory networks as computing agents in silico and in vivo. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 320–335. Springer, Heidelberg (2007)

11. Hucka, M., et al.: The systems biology markup language SBML: A medium for representation and exchange of biochemical network models. Bioinformatics 19, 524–531 (2003)
12. Lenser, T., Hinze, T., Ibrahim, B., Dittrich, P.: Towards evolutionary network reconstruction tools for systems biology. In: Marchiori, E., Moore, J.H., Rajapakse, J.C. (eds.) EvoBIO 2007. LNCS, vol. 4447, pp. 132–142. Springer, Heidelberg (2007)
13. Magnasco, M.O.: Chemical kinetics is Turing universal. Physical Review Letters 78, 1190–1193 (1997)
14. Manca, V.: Metabolic P systems for biomolecular dynamics. Progress in Natural Sciences 17, 384–391 (2006)
15. Nepomuceno, I., et al.: A tool for using the SBML format to represent P systems which model biological reaction networks. In: Proc. 3rd Brainstorming Week on Membrane Computing, Fenix Editora, Sevilla, pp. 219–228 (2005)
16. Păun, G.: Computing with membranes. J. Comp. Syst. Sci. 61, 108–143 (2000)
17. Păun, G.: Membrane Computing: An Introduction. Springer, Heidelberg (2002)
18. Pescini, D., et al.: Investigating local evolutions in dynamical probabilistic P systems. In: Ciobanu, G., et al. (eds.) Proc. First Intern. Workshop on Theory and Application of P Systems, pp. 275–288 (2005)
19. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer, Heidelberg (1997)
20. Suzuki, Y., Tanaka, H.: Symbolic chemical system based on abstract rewriting system and its behavior pattern. Artificial Life and Robotics 1, 211–219 (1997)
21. Zhabotinsky, A.M.: 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], and Alfonso Rodríguez-Patón[2]

[1] Department of Computer Science/IfM
Louisiana Tech University, P.O. Box 10348, Ruston, LA 71272, USA
{johnjack,apaun}@latech.edu
[2] Departamento de Inteligencia Artificial, Facultad de Informática
Universidad Politécnica de Madrid,
Campus de Montegancedo S/N, Boadilla del Monte, 28660 Madrid, Spain
arpaton@fi.upm.es
[3] Bioinformatics Department, National Institute of Research and Development for Biological Sciences,
Splaiul Independenţei, Nr. 296, Sector 6, Bucharest, Romania

Abstract. 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,

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 246–259, 2009.

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 [20], using the simulation technique reported in [22]. 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 [44].

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 [14] 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 [15,41,43]. Understanding the mechanisms behind HIV-1 latency is a focal point for current AIDS-related research (for a recent review on latency see [18]).

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) [35,47,48,50], 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 [15]. 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 [22] from simulating the Fas-mediated apoptotic signaling cascade, based on information for the Jurkat T cell line [20]. In [22], 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 [20]. 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 [22], 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 Computing ([33]) 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 [22] 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 [16,17], 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

3.1 Fas-Mediated Apoptosis

We have explored the literature pertaining to the effects of HIV-1 proteins on apoptosis: see [2,38,40] 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 [21] and [36], understanding the complex signaling cascade of Fas-mediated apoptosis can be beneficial in developing remedies for cancer and autoimmune disorders.

In [39], 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 [27].

In [22], 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.

3.2 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 [8,45]. 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. 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 [24]. The HIV_{cDNA} is then integrated into the host's genome with the help of the viral integrase approximately one hour later [13]. These rules are shown in Table 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 [25,29]. 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} [25]. 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, [25] 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 1.

Multiply spliced (MS) HIV-1 mRNAs – responsible for Tat/Rev protein creation – are detectable in resting CD4+ T cells [26]. However, due to the inefficient export of the 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,

Table 1. A partial list of the reactions involving effects of HIV-1 proteins, which were added to the existing Fas model [20,22]. See the WMC9 pre-proceedings version of this paper for the complete list of reactions.

Reaction	**Reaction Rate**
1: HIV_{RNA} + RT → HIV_{cDNA} + RT	k_{21}
2: HIV_{cDNA} → HIV_{cDNA} (nuclear import)	k_{22}
3: HIV_{cDNA} → HIV_{LTR}	k_{22}
4: NFAT → NFAT (nuclear import)	k_{23}
5: CDK9 → CDK9 (nuclear import)	k_{24}
6: CyclinT1 + CDK9 → PTEFb	k_{25}
7: NFAT + HIV_{LTR} → HIV_{LTR}:NFAT	k_{26}
8: HIV_{LTR}:NFAT + Tat → HIV_{LTR}:NFAT:Tat	k_{27}
9: HIV_{LTR}:NFAT:Tat + PTEFb → HIV_{LTR}:NFAT:Tat:PTEFb	k_{28}
10: HIV_{LTR} → HIV_{LTR} + mRNA_{Tat}	k_{29}
11: HIV_{LTR} → HIV_{LTR} + mRNA_{Vpr}	k_{29}
12: HIV_{LTR} → HIV_{LTR} + $\text{mRNA}_{HIV_{pr}}$	k_{29}
13: HIV_{LTR} → HIV_{LTR} + mRNA_{Nef}	k_{29}
14: HIV_{LTR}:NFAT → HIV_{LTR}:NFAT + mRNA_{Tat}	k_{30}
15: HIV_{LTR}:NFAT → HIV_{LTR}:NFAT + mRNA_{Vpr}	k_{30}
16: HIV_{LTR}:NFAT → HIV_{LTR}:NFAT + $\text{mRNA}_{HIV_{pr}}$	k_{30}
17: HIV_{LTR}:NFAT → HIV_{LTR}:NFAT + mRNA_{Nef}	k_{30}
18: HIV_{LTR}:NFAT:Tat → HIV_{LTR}:NFAT:Tat + mRNA_{Tat}	k_{31}
19: HIV_{LTR}:NFAT:Tat → HIV_{LTR}:NFAT:Tat + mRNA_{Vpr}	k_{31}
20: HIV_{LTR}:NFAT:Tat → HIV_{LTR}:NFAT:Tat + $\text{mRNA}_{HIV_{pr}}$	k_{31}
21: HIV_{LTR}:NFAT:Tat → HIV_{LTR}:NFAT:Tat + mRNA_{Nef}	k_{31}
26: mRNA_{Tat} → mRNA_{Tat} (nuclear export)	k_{33}
27: mRNA_{Nef} → mRNA_{Nef} (nuclear export)	k_{33}
28: mRNA_{Vpr} → mRNA_{Vpr} (nuclear export)	k_{33}
29: $\text{mRNA}_{HIV_{pr}}$ → $\text{mRNA}_{HIV_{pr}}$ (nuclear export)	k_{33}
30: mRNA_{Tat} → mRNA_{Tat} + Tat	k_{34}
31: mRNA_{Nef} → mRNA_{Nef} + Nef	k_{34}
32: mRNA_{Vpr} → mRNA_{Vpr} + Vpr	k_{34}
33: $\text{mRNA}_{HIV_{pr}}$ → $\text{mRNA}_{HIV_{pr}}$ + HIV_{pr}	k_{34}
34: mRNA_{Tat} → degraded	k_{35}
35: mRNA_{Nef} → degraded	k_{35}
36: mRNA_{Vpr} → degraded	k_{35}
37: $\text{mRNA}_{HIV_{pr}}$ → degraded	k_{35}
38: Tat ⇌ Tat (nuclear import/export)	k_{36f}, k_{35r}
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}

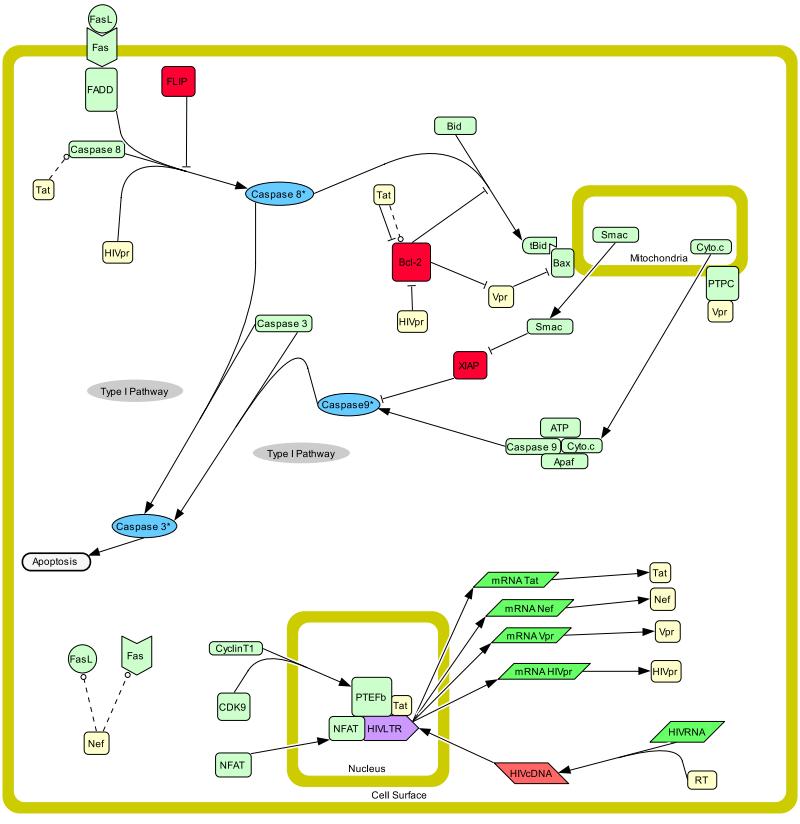


Fig. 1. A picture of the model for HIV-1 protein effects on Fas signaling. The activation of Caspase 3 is the end of the signaling cascade – irrevocably leads to cell death. The type I pathway involves direct activation of Caspase 3 by Caspase 8*. The type II pathway requires signal amplification by 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.

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.

In an inactivated cell, CyclinT1 and CDK9 are sequestered in the cytoplasm [30]. 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 1. The transcription, translocation, and translation rules involving HIV-1 mRNA molecules are also given in Table 1.

3.3 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 [40]. 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, [14] 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 [4,46], which may effect the cell through autocrine signaling. The anti- and pro-apoptotic rules for Tat are found in Table 1.

The HIV-1 Vpr has been shown to both enhance and inhibit the Fas signaling cascade. Upon infection, the ∼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 [23]. 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 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 [42], while it also cleaves Caspase 8 [31] into active form. Both rules are pro-apoptotic and are in Table 1.

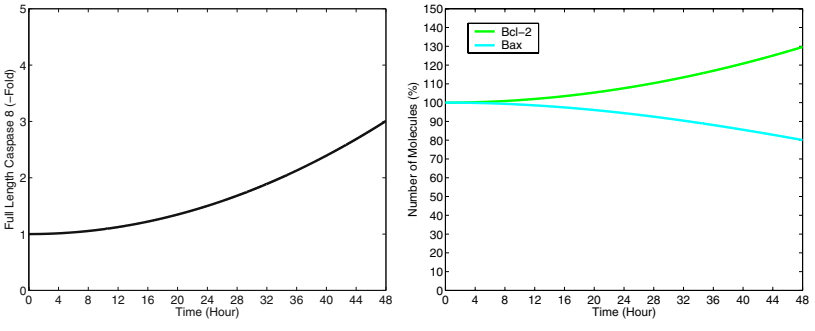
Finally, we define two pro-apoptotic rules for the Nef protein. Zauli et al. discovered in [49] that Nef can play a role in cell death by upregulating Fas receptor and Fas ligand on the cell surface. Upregulating the receptor sites of Fas on the cell surface prepares the cell for ligand binding, and 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 1.

4 Results

We added all of the rules from Table 1 to the Fas model described in [20,22]. 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. 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. 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].



(a) (b)

Fig. 2. (a) Tat protein upregulates Caspase 8 levels by three-fold. (b) Vpr upregulates Bcl-2 and downregulates Bax by 30% and 20%, resp.

We will next consider the activation of Caspase 3. In Fig. 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. 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.

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) 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

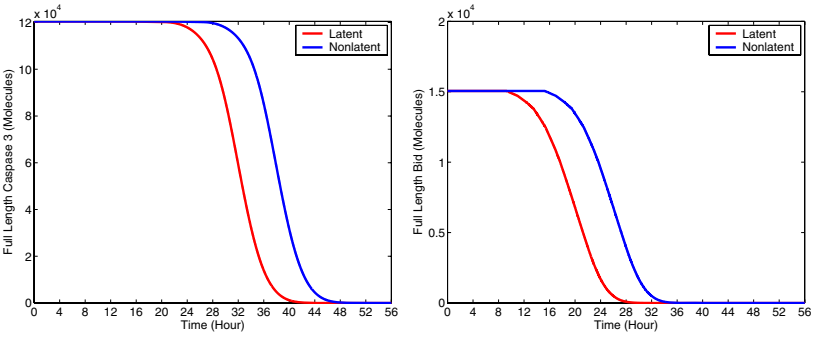


Fig. 3. (a)Total reduction of full length Caspase 3 is seen after $\sim$ 40 hours in the *latent* model, whereas the *nonlatent* model takes $\sim$ 47 hours. (b) The decline of Bid through interactions with Caspase 8, leading to a rise in tBid.

with Caspase 9*, after signal amplification through the mitochondria. As discussed in [20,22], 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.

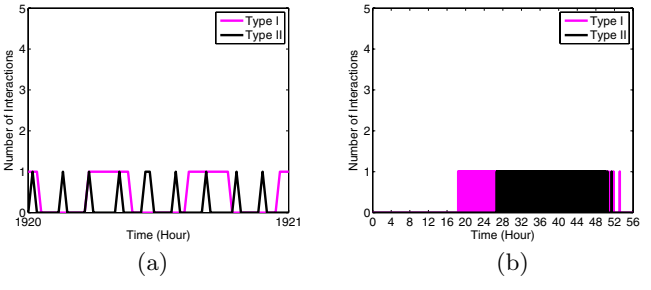
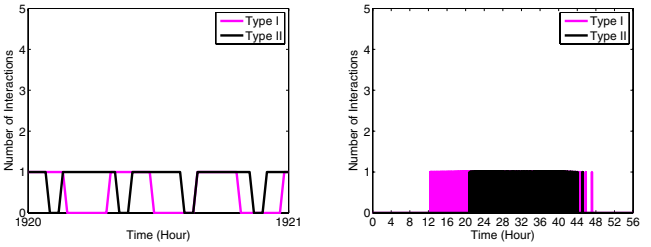


Fig. 4. These graphs show the type I and type II pathways working together to activate Caspase 3 during the *nonlatent* simulation. The type I interactions are active Caspase 8 binding with Caspase 3, and the type II interactions are Caspase 9 binding with Caspase 3. (a) The overall picture for the whole three days of simulation. (b) One minute from the simulation (from 32 hours to 32 hours and 1 minute) illustrates the rate of interactions.

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. 5), we see type I interactions first occur about 12 hours into the simulation, while type II molecular binding occurs after 21 hours.

Although Fig. 4(b) and Fig. 5(b) imply type I interactions occur more frequently than type II, it must be noted that, due to the kinetics governing these



(a) (b)

Fig. 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.

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 [14]. 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 [2,38,40]. 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 [19], increased Caspase 8 activation [1], and upregulation of Fas [32], FasL and Bax [40].

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 [28]. Endocytosed Tat can upregulate levels of Caspase 8 [4] and increase expression of the Fas ligand [40], 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* [40].

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).

References

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, 1636–1642 (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, 1284–1290 (1995)

10. Chun, T.W., et al.: Quantification of latent tissue reservoirs and total body viral load in HIV-1 infection. Nature 387, 183–188 (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. Farnet, C., Haseltine, W.: Determination of viral proteins present in the human immunodeficiency virus type 1 preintegration complex. J. Vir. 65, 1910–1915 (1991)
14. 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)
15. 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, 512–517 (1999)
16. Gillespie, D.T.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. J. Comp. Phy. 22, 403–434 (1976)
17. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. The J. Phy. Chem. 81, 2340–2361 (1997)
18. Han, T., et al.: Experimental approaches to the study of HIV-1 latency. Microbio. 5, 95–106 (2007)
19. 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)
20. 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, 6235–6237 (2005)
21. Igney, F., Krammer, P.: Death and anti-death: tumour resistance to apoptosis. Nature Rev. Can. 2, 277–288 (2002)
22. Jack, J., et al.: Discrete nondeterministic modeling of the Fas pathway. Intern. J. Found. Computer Sci. (accepted, 2008)
23. 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)
24. 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)
25. 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)
26. Lassen, K., et al.: Nuclear retention of multiply spliced HIV-1 RNA in resting CD4+ T cells. PLoS Pathogens 2, 650–661 (2006)
27. 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)
28. McCloskey, T., et al.: Dual role of HIV Tat in regulation of apoptosis in T cells. J. Immun. 158, 1014–1019 (1997)
29. Nabel, G., Baltimore, D.: An inducible transcription factor activates expression of human immunodeficiency virus in T cells. Nature 344, 711–713 (1987)
30. 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)

31. 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)
32. 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)
33. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
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, 8518–8531 (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. Scaffidi, C., et al.: Two CD95 (APO-1/Fas) signaling pathways. EMBO J. 17, 1675–1687 (1998)
40. Selliah, N., Finkel, T.: Biochemical mechanisms of HIV induced T cell apoptosis. Cell Death Diff. 8, 127–136 (2001)
41. 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, 727–728 (2003)
42. Strack, L., 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)
43. 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, 4819–4824 (2003)
44. World Health Organization 2007 AIDS Epidemic Update, `http://www.who.int/hiv/en/`
45. Wyatt, R., Sodroski, J.: The HIV-1 envelope glycoproteins: fusogens, antigens, and immunogens. Science 280, 1884–1888 (1998)
46. 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)
47. Zack, J.A., et al.: HIV-1 entry into quiescent primary lymphocytes: molecular analysis reveals a labile, latent viral structure. Cell 61, 213–222 (1990)
48. 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, 1717–1725 (1992)
49. 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)
50. Zhou, Y., et al.: Kinetics of human immunodeficiency virus type 1 decay following entry into resting CD4+ T cells. J. Vir. 79, 2199–2210 (2005)

Transforming State-Based Models to P Systems Models in Practice

Petros Kefalas[1], Ioanna Stamatopoulou[2],
George Eleftherakis[1], and Marian Gheorghe[3]

[1] Department of Computer Science, CITY College, Thessaloniki, Greece
{kefalas,eleftherakis}@city.academic.gr
[2] South-East European Research Centre, Thessaloniki, Greece
istamatopoulou@seerc.org
[3] Department of Computer Science, University of Sheffield, UK
M.Gheorghe@dcs.shef.ac.uk

Abstract. 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 modeling reactive systems [7]. In particular, X-machines (XMs) possess an intuitive modeling 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 modeling 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 modeling message passing means and computation that demonstrate the feasibility of scaling up models [8]. However, they do suffer from a major drawback: the organizational 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, reorganization is an important feature that should be addressed in a model. In

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 260–273, 2009.

this context, by reorganization 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 modeling 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 modeling behaviors of agents because simple objects and rewriting rules over those objects are not sufficient. But they do deal with reorganization 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 modeling aspects.

A brief comparison between Communicating X-machines and Population P Systems is shown in Table 1. Their complementarity has led to the successful integration of the two methods [14].

Table 1. Comparison of features of X-machines and Population P Systems with respect to modeling

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	√	√

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 modelers, we would rarely use PPS for modeling the behavior 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 modeling agent behaviors, 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 organized 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 modeling 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. 1.

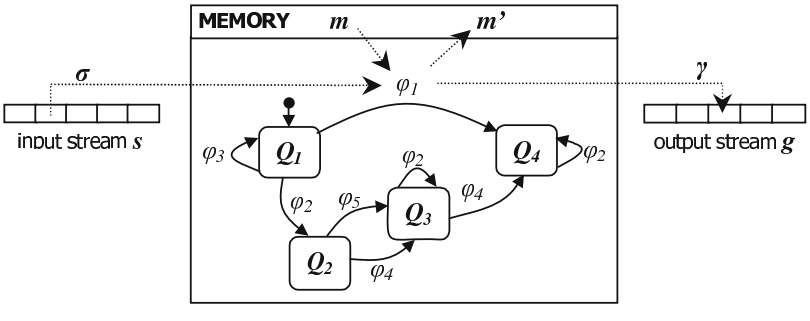


Fig. 1. An abstract X-machine

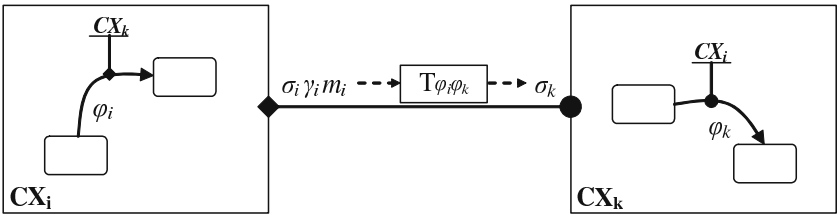


Fig. 2. An abstract example of two Communicating X-machines

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 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 while a complete definition of CXMs can be found in [8].

The X-Machine Description Language (XMDL) was developed to assist with the modeling 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

Table 2. Main constructs of XMDL (words in upright font are XMDL keywords)

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.

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 modeler 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 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 3).

Table 3. Main constructs of XMDL-c (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.

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. 3.

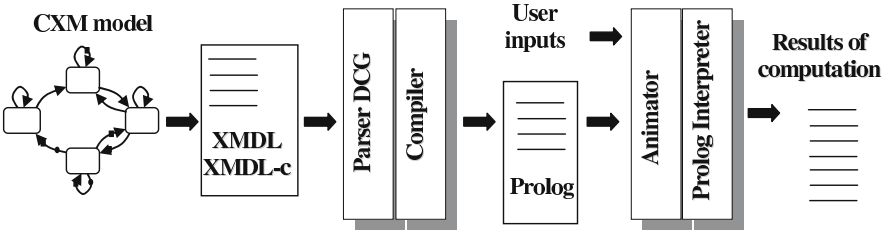


Fig. 3. The X-System

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 neighboring 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].

Table 4. Main constructs of PPSDL (words in upright font are PPSDL keywords)

PPS	**PPSDL syntax**
V	`#object` (obj_name) `=(`$type\_name_1, \ldots$`)`
K	`#cell_types =` $(cell\_type_1, \ldots, cell\_type_m)$
	`#cell_names =` $\{cell\_name_1, \ldots, cell\_name_n\}$
γ	`#graph` $graph\_name = \{(cell\_name_1, cell\_name_2), \ldots\}$
w_E	`#env_objects =` $\{obj_1, obj_2, \ldots\}$
$C_i = (w_i, t_i)$	`#cell` $(cell\_name_1)$ `:` $([obj_1, obj_2, \ldots], cell\_type)$.
$(a \rightarrow b)_t$	`#transformation_rule` $rule\_name$ `([`$obj_1, \ldots$`] -> [`$obj_i, \ldots$`]):`$cell\_type_1$`,` `if` $cond_1$ `(and|or)` $cond_2, \ldots$, `where` $informative\_expression$.
$(a; b, in)_t$, $(a; b, enter)_t$	`#communication_in_rule` $rule\_name$ `( when [`$obj_1, \ldots$`] receive [`$obj_i, \ldots$`]` `(from_cell | from_environment )) :` $cell\_type$.
$(a, exit, b)_t$	`#communication_exit_rule` $rule\_name$ `( when [`$obj_i, \ldots$`] ( send_to_cell` `| send_to_environment) [`$obj_j, \ldots$`] ) :` $cell\_type$.
$(a)_t \rightarrow (b)_p$	`#differentiation_rule` $rule\_name$ `([`$obj_1, \ldots$`]) :` $cell\_type_1$ `-> ([`$obj_i, \ldots$`]) :` $cell\_type_2$.
$(a)_t \rightarrow$ $(b)_t(c)_t$	`#division_rule` $rule\_name$ `([`$obj_1, \ldots$`]) :` $cell\_type$ `->([`$obj_i, \ldots$`]) :` $cell\_type$ `([`$obj_j, \ldots$`]) :` $cell\_type$.
$(a)_t \rightarrow \dagger$	`#death_rule` $rule\_name$ `([`$obj_1, \ldots$`]) :` $cell\_type$ `->+`
$(i, x_1; x_2, j)$	`#bond_making_rule` $rule\_name$ `(when [`$obj_1, \ldots$`] :` $cell\_type$ `and [`$obj_1, \ldots$`] :` $cell\_type$ `).`
	`#define` $identifier$ `=` $user\ defined$ `|` $set\ operations$ `|` $built\text{-}in\ type$ `|` $tuple$

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 modeling is the ability to associate objects with types, by combining built-in types with user-defined types. Therefore, objects in cells are characterized by a type identifier; we use the notation `type_identifier*(value)` to denote objects. Table 4 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. 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.

4 Transforming XMDL to PPSDL

Recently, some basic principles for transforming CXM to PPS have been reported [10] and are briefly presented in Table 5. 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:

Table 5. Principles of transforming a CXM to a PPS

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 $incoming \in \Sigma$, $T(\sigma, m, \gamma) = outgoing$	$\varphi_{\sigma,m}$: $((state : q)$ $(memory : m)$ $(message : incoming)$ $\rightarrow (state : q')$ $(memory : m')$ $(output : \gamma)$ $(message : outgoing))_t$

```
 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.

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 behavior 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 behavior 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. 4 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.

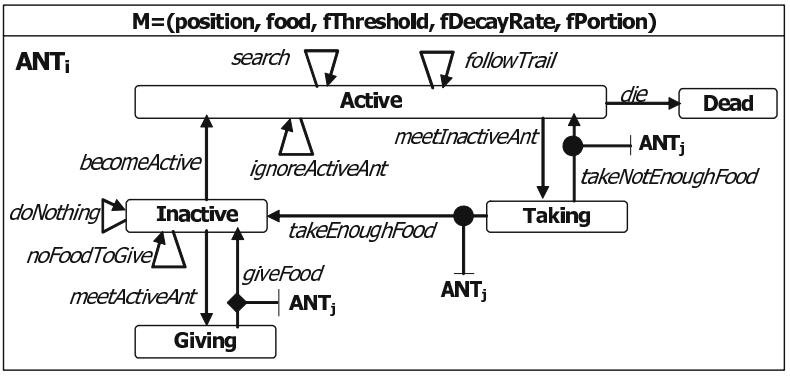


Fig. 4. The Pharaoh ant X-machine model

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 modeling 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]]
...
```

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.

References

1. Bernardini, F., Gheorghe, M.: Population P systems. J. Universal Computer Sci. 10, 509–539 (2004)
2. Eleftherakis, G.: Formal Verification of X-machine Models: Towards Formal Development of Computer-based Systems. PhD thesis, Department of Computer Science, University of Sheffield (2003)
3. Gheorghe, M., Stamatopoulou, I., Holcombe, M., Kefalas, P.: Modelling dynamically organised colonies of bio-entities. In: Banâtre, J.-P., Fradet, P., Giavitto, J.-L., Michel, O. (eds.) UPP 2004. LNCS, vol. 3566, pp. 207–224. Springer, Heidelberg (2005)
4. Holcombe, M., Ipate, F.: Correct Systems: Building a Business Process Solution. Springer, Heidelberg (1998)
5. Kapeti, E., Kefalas, P.: A design language and tool for X-machines specification. In: Fotiadis, D.I., Spyropoulos, S.D. (eds.) Advances in Informatics, pp. 134–145. World Scientific Publishing Company, Singapore (2000)
6. Kefalas, P.: XMDL User Manual. CITY College, Thessaloniki, Greece (2000)
7. Kefalas, P.: Formal modelling of reactive agents as an aggregation of simple behaviours. In: Vlahavas, I.P., Spyropoulos, C.D. (eds.) SETN 2002. LNCS (LNAI), vol. 2308, pp. 461–472. Springer, Heidelberg (2002)
8. Kefalas, P., Eleftherakis, G., Kehris, E.: Modular modelling of large-scale systems using communicating X-machines. In: Manolopoulos, Y., Evripidou, S. (eds.) Proc. 8th Panhellenic Conference in Informatics, pp. 20–29. Livanis Publishing Company (2001)
9. Kefalas, P., Eleftherakis, G., Sotiriadou, A.: Developing tools for formal methods. In: Proc. 9th Panhellenic Conference in Informatics, pp. 625–639 (2003)

10. Kefalas, P., Stamatopoulou, I., Gheorghe, M.: Principles of transforming communicating X-machines to population P systems. In: Vaszil, G. (ed.) Proc. Intern. Workshop on Automata for Cellular and Molecular Computing (ACMC 2007), pp. 76–89 (2007)
11. Păun, G.: Computing with membranes. J. Computer and System Sci. 61, 108–143 (2000)
12. Stamatopoulou, I., Gheorghe, M., Kefalas, P.: Modelling dynamic organization of biology-inspired multi-agent systems with communicating X-machines and population P systems. In: Mauri, G., Păun, G., Jesús Pérez-Jímenez, M., Rozenberg, G., Salomaa, A. (eds.) WMC 2004. LNCS, vol. 3365, pp. 389–403. Springer, Heidelberg (2005)
13. Stamatopoulou, I., Kefalas, P., Eleftherakis, G., Gheorghe, M.: A modelling language and tool for population P systems. In: Bozanis, P., Houstis, E.N. (eds.) PCI 2005. LNCS, vol. 3746. Springer, Heidelberg (2005)
14. Stamatopoulou, I., Kefalas, P., Gheorghe, M.: OPERAS: a formal framework for multi-agent systems and its application to swarm-based systems. In: Artikis, A., O'Hare, G.M.P., Stathis, K., Vouros, G. (eds.) ESAW 2007. LNCS, vol. 4995, pp. 208–223. Springer, Heidelberg (2007)
15. The P Systems webpage, `http://ppage.psystems.eu/index.php/Software`
16. Thomson, C., Holcombe, M.: Using a formal method to model software design in XP projects. In: Eleftherakis, G. (ed.) Proc. 2nd South-East European Workshop on Formal Methods, pp. 74–88 (2005)
17. Transforming state-based models to P Systems models in practice. Support documentation and demos,
`www.city.academic.gr/csd/kefalas/XMDLtoPPSDL/index.html`

How Redundant Is Your Universal Computation Device?

Alberto Leporati, Claudio Zandron, and Giancarlo Mauri

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano – Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
{leporati,zandron,mauri}@disco.unimib.it

Abstract. 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* of M (under the encoding $\mathcal{C}$) as the length of $\mathcal{C}(M)$. The description size 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 $\mathcal{M}$. In this paper we propose the description size as a complexity measure that allows to compare different computational models. We compute upper bounds to the description size of deterministic register machines, Turing machines, spiking neural P systems and UREM P systems. By comparing these sizes, we provide a first partial answer to the following intriguing question: what is the minimal (description) size of a universal computation device?

1 Introduction

Looking for small universal computing devices is a natural and well investigated topic in computer science: see e.g., [24,12,20,21,22,9,10] for classic computational models, [23,6,4] for tissue and symport/antiport P systems, [26,3] for cellular automata, and [7,17] for spiking neural P systems.

A related question that we investigate in this paper is: What is the size of the *smallest* among all possible universal computation devices? Of course we must agree on the meaning of the term "size", since 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 $\mathcal{C} : \mathcal{M} \to \{0,1\}^*$ 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

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 274–291, 2009.

devices contained in $\mathcal{M}$. Then, we start a quest for the shortest possible bit string that describes a universal computation device: that is, we look for a computational model $\mathcal{M}$ and an encoding function $\mathcal{C} : \mathcal{M} \to \{0,1\}^*$ 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, Turing machines, spiking neural P systems [8] and UREM P systems [5], 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 other 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 [18,19], whose structure and behavior are inspired from the functioning of living cells; the amount of theoretical results and applications reported in the bibliography of [27] 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. Since several variants of Turing machines have been defined in the literature, we will later refer the reader to the bibliography for the details on the these machines. In Section 3 we will define and compute the description size of *randomly chosen* instances of all these models. We will also consider a small 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 and we propose some directions for future research.

2 Some (Universal) Models of Computation

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 labeled 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, labeled 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 (labeled with 0), and terminate when the instruction currently executed tries to jump to label m.

For a formal definition of *configurations* and *computations* of M we refer the reader to [5]. Here we just recall that deterministic register machines provide a simple universal computational model, as stated in [5, Proposition 1].

2.2 Spiking Neural P Systems

Spiking neural P systems (SN P systems, for short) have been introduced in [8] 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.

Formally, a *spiking neural membrane system* (SN P system, for short) of degree $m \geq 1$, as defined in [7] 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;
      (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. 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.

Extended rules provide a common generalization of firing 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.

We refer the reader to [7] for a formal definition of *configurations* and *computations*; here we just recall that (many variants of) SN P systems have proven to be universal. In what follows we will use the two small deterministic universal SN P systems which are defined in [17]: one of them uses 84 neurons, each one containing only standard rules; the other uses 49 neurons, but in this case extended rules are needed.

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 [5] 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 [5] 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 labeled 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 $0, \ldots, d$. 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 of R_i has the form $(\alpha_i : a, \Delta e, b)$, 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.

A computation step is performed by *non-deterministically* choosing one rule from some R_i and applying it (hence in a *sequential* way, as opposed to the maximally parallel way 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, a sort of local priorities is assumed: if there are two or more applicable rules in membrane i, then one of the rules with $\max |\Delta e|$ has to be used.

If we consider the distribution of energy values among some predefined membranes as the input to be processed and the resulting output (a non–halting computation does not produce a result) we obtain a universal model of computation, as proved in [5, Theorem 1]. The proof is obtained by simulating deterministic register machines by *deterministic* UREM P systems which contain

one elementary membrane into the skin for each register of the simulated machine. The contents of each register are expressed as the energy value assigned to the corresponding membrane. 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 *necessary* to correctly simulate a decrement instruction, and hence to reach the computational power of Turing machines, as proved in [5, Theorem 2].

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 [2]. 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 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.

We can look at any computational model as a family of computation devices, whose size depends upon a predefined 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 subclass of $\mathcal{M}$ that is composed of those devices $M \in \mathcal{M}$ for which the parameters have the indicated values $n_1, \ldots, n_k$. An encoding function $\mathcal{C} : \mathcal{M} \to \{0,1\}^*$ can thus be viewed as a family $\{\mathcal{C}(n_1, \ldots, n_k)\}_{n_1,\ldots,n_k}$, where the function $\mathcal{C}(n_1, \ldots, n_k)$ encodes any instance $M \in M(n_1, \ldots, n_k)$. Note that the values $n_1, \ldots, n_k$ need not to be encoded, since they can be determined by the sub-function of $\mathcal{C}$ we are using. In what follows we will propose specific encodings $\mathcal{C}(n_1, \ldots, n_k)$ for each of the computational models considered in this paper, both for generic and for specific values of $n_1, \ldots, n_k$; with a little abuse of notation, we will sometimes indicate the functions $\mathcal{C}(n_1, \ldots, n_k)$ as the encodings (that is, $\mathcal{C}$) of our models.

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 computation devices $\mathcal{M}$ and $\mathcal{M}'$ (with possibly $\mathcal{M} = \mathcal{M}'$), 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 $R_{\mathcal{M},\mathcal{C}}(M') = \frac{ds_{\mathcal{C}}(M')}{ds_{\mathcal{C}}(\mathcal{M})}$. Similarly, we define the redundancy of a computational model $\mathcal{M}'$ (with respect to $\mathcal{M}$ and $\mathcal{C}$) as $R_{\mathcal{M},\mathcal{C}}(\mathcal{M}') = \frac{ds_{\mathcal{C}}(\mathcal{M}')}{ds_{\mathcal{C}}(\mathcal{M})}$. 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 size 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 each of the classes mentioned in section 2: deterministic register machines, Turing machines, SN P systems and UREM P systems (each with respect to an appropriate predefined encoding). Then, we will also compute the description size of a small universal device taken from each of these classes, 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 $\lfloor \log_2 n \rfloor + 1$ of bits which are needed to represent n in binary form.

3.1 Deterministic Register Machines

Denoted by $\mathrm{DRM}(n, m)$ the subclass of register machines that have n registers and programs composed of m instructions, we can express the class DRM of deterministic register machines as: $\mathrm{DRM} = \bigcup_{n \geq 1, m \geq 0} \mathrm{DRM}(n, m)$. Let us describe a function $\mathcal{C}(n, m)$ that encodes any machine from the subclass $\mathrm{DRM}(n, m)$.

Let $M \in \mathrm{DRM}(n, m)$, and let P denote its program. To describe each 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. A simple encoding of M 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.

If M is a *randomly chosen* (and thus, possibly, non–universal) machine, then 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 [9], several universal register machines are described and investigated. In particular, the *small* universal register machine illustrated in Figure 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

$0 : (DEC(1), 1, 2)$ $1 : (INC(7), 0)$
$2 : (INC(6), 3)$ $3 : (DEC(5), 2, 4)$
$4 : (DEC(6), 5, 3)$ $5 : (INC(5), 6)$
$6 : (DEC(7), 7, 8)$ $7 : (INC(1), 4)$
$8 : (DEC(6), 9, 0)$ $9 : (INC(6), 10)$
$10 : (DEC(4), 0, 11)$ $11 : (DEC(5), 12, 13)$
$12 : (DEC(5), 14, 15)$ $13 : (DEC(2), 18, 19)$
$14 : (DEC(5), 16, 17)$ $15 : (DEC(3), 18, 20)$
$16 : (INC(4), 11)$ $17 : (INC(2), 21)$
$18 : (DEC(4), 0, 22)$ $19 : (DEC(0), 0, 18)$
$20 : (INC(0), 0)$ $21 : (INC(3), 18)$

Fig. 1. The small universal deterministic register machine defined in [9]

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. This means, of course, that the encoding algorithm will have to produce a compressed representation of M, that will be decompressed by the decoding algorithm. Many compression algorithms exist, that yield different results. For simplicity 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.

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 modeled the output of the encoding algorithm as a memoryless information source is questionable. In fact, when we encounter a 0 at the beginning of a new block that encodes an instruction of the register machine then it is clear 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 depends 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.

3.2 Turing Machines

Let $\mathrm{TM}(n, m)$ be the class of Turing machines with n symbols and m states. Several definitions of Turing machines, all equivalent from the computational power point of view, have been given in the literature. Here we refer to the traditional (standard) one, with a bi-infinite tape and one read/write head. We refer to [25] for a formal definition of Turing machines, configurations and computations.

Several authors have investigated small universal Turing machines. For example, Rogozhin [22] constructed small universal machines in the classes $\mathrm{TM}(5, 5)$, $\mathrm{TM}(6, 4)$, $\mathrm{TM}(10, 3)$ and $\mathrm{TM}(18, 2)$, Kudlek and Rogozhin [11] constructed a machine in $\mathrm{TM}(9, 3)$, and Baiocchi [1] constructed machines in $\mathrm{TM}(2, 19)$ and $\mathrm{TM}(4, 7)$. The smallest machine, both in terms of number of instructions (22) and number of symbol–state pairs (24) is Rogozhin's machine in $\mathrm{TM}(6, 4)$. Let us note in passing that, by slightly modifying the definition of Turing machines (and, consequently, the notion of universality) it is possible to obtain even smaller machines [15,16,3,26]. Concerning traditional machines, assuming that a single instruction is reserved for halting, it is known that there are no universal machines in $\mathrm{TM}(2, 2)$, $\mathrm{TM}(2, 3)$, $\mathrm{TM}(3, 2)$, $\mathrm{TM}(n, 1)$ and $\mathrm{TM}(1, n)$ for $n \geq 1$. See [14] for further details and references.

Given $M \in \mathrm{TM}(n, m)$, we need $\lg n$ and $\lg m$ bits to represent each symbol and each state, respectively. The read/write head can only move to the cell on its left, to the cell on its right, or not move; two bits are thus needed to represent the head movement. A simple encoding of M is a sequence of blocks of $2 \lg n + 2 \lg m + 2$

bits, each encoding an instruction of the kind *(current state, current symbol, new state, new symbol, head movement)*. The maximum number of blocks is $n \cdot m$. By writing the 2 head movement bits at the beginning of each block, we can encode the empty instruction (for those symbol–state pairs that do not have an instruction) as the unused head movement 2-bit configuration. In this way, the description size (under this encoding $\mathcal{C}$) of any Turing machine $M \in \mathrm{TM}(n, m)$ having $k \leq n \cdot m$ instructions will be $ds_{\mathcal{C}}(M) = k \cdot (2 \lg n + 2 \lg m + 2) + (n \cdot m - k) \cdot 2$ bits.

Using this encoding, each non-empty instruction of Rogozhin's 6-symbol 4-state 22-instruction universal machine requires 12 bits. Hence the description size of the machine is 268 bits, which is an upper bound to the description size $ds(\mathrm{TM})$ of the class of Turing machines. To limit the length of the paper, we do not show the explicit bit string that encodes Rogozhin's machine; we just note that its size is just a little bit higher than the (uncompressed) size of the smallest currently known register machine.

3.3 Spiking Neural P Systems

Let $\mathrm{SNP}(m, R, C, D)$ denote the class of SN P systems having degree m and a total number R of rules, where each rule consumes a maximum number C of spikes and has a maximum delay D.

Let $\Pi \in \mathrm{SNP}(m, R, C, D)$. In order to describe the synapse graph of Π (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 [17] we note that the expressions $a, a^2, a^3, a(aa)^+$ and $a(aa)^*$ suffice to reach computational completeness, 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. 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 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 it.

A simple encoding of Π 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 (possibly zero), we will put an additional bit equal to 1 in front of the encoding 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 $\Pi \in \mathrm{SNP}(m, R, C, D)$ 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 [17], 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 [17] for details); the modified version of the register machine has 9 registers, 24 instructions 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 [17]. 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 $ds(\text{SNP})$, 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 $ds(\text{DRM}) = 263$ and $ds(\text{SNP}) = 7773$, an approximated value of the *redundancy* of spiking neural P systems with respect to deterministic register machines, is:

$$R_{\text{DRM}}(\text{SNP}) = \frac{ds(\text{SNP})}{ds(\text{DRM})} = \frac{7773}{263} \approx 29.56$$

On the other hand, by assuming $ds(\text{DRM}) = 258$ (that results from the computation of the entropy of the string that encodes the universal deterministic register machine depicted in Figure 1) the redundancy becomes $R_{\text{DRM}}(\text{SNP}) = \frac{ds(\text{SNP})}{ds(\text{DRM})} = \frac{7773}{258} \approx 30.13$. 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.

In [17] 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 also need 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 [17], 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 $ds(\text{SNP})$ and, assuming $ds(\text{DRM}) = 263$ or $ds(\text{DRM}) = 258$, we obtain

$$R_{\text{DRM}}(\text{SNP}) = \frac{4361}{263} \approx 16.58 \qquad \text{and} \qquad R_{\text{DRM}}(\text{SNP}) = \frac{4361}{258} \approx 16.90$$

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.

3.4 UREM P Systems

Let UREM denote the class of UREM P systems. As proved in [5], 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 when 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, we can define the subclass $\text{UREM}(n, m, R)$ of UREM P systems having R rules, an alphabet of m symbols, and n elementary membranes contained into the skin. Given $\Pi \in \text{UREM}(n, m, R)$, 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 append 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. A simple encoding of Π 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 stated in Section 2.3, to simulate the instructions of the register machine we need the objects p_j and $\widetilde{p}_j$, for all $j \in \{0, 1, \ldots, 21\}$ (see [5] for details), 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, reported in Figure 2.

$0 : (in_1 : p_0, 0, \widetilde{p}_0),\ (out_1 : \widetilde{p}_0, -1, p_1),\ (out_1 : \widetilde{p}_0, 0, p_2)$
$1 : (in_7 : p_1, 1, \widetilde{p}_1),\ (out_7 : \widetilde{p}_1, 0, p_0)$
$2 : (in_6 : p_2, 1, \widetilde{p}_2),\ (out_6 : \widetilde{p}_2, 0, p_3)$
$3 : (in_5 : p_3, 0, \widetilde{p}_3),\ (out_5 : \widetilde{p}_3, -1, p_2),\ (out_5 : \widetilde{p}_3, 0, p_4)$
$4 : (in_6 : p_4, 0, \widetilde{p}_4),\ (out_6 : \widetilde{p}_4, -1, p_5),\ (out_6 : \widetilde{p}_4, 0, p_3)$
$5 : (in_5 : p_5, 1, \widetilde{p}_5),\ (out_5 : \widetilde{p}_5, 0, p_6)$
$6 : (in_7 : p_6, 0, \widetilde{p}_6),\ (out_7 : \widetilde{p}_6, -1, p_7),\ (out_7 : \widetilde{p}_6, 0, p_8)$
$7 : (in_1 : p_7, 1, \widetilde{p}_7),\ (out_1 : \widetilde{p}_7, 0, p_4)$
$8 : (in_6 : p_8, 0, \widetilde{p}_8),\ (out_6 : \widetilde{p}_8, -1, p_9),\ (out_6 : \widetilde{p}_8, 0, p_0)$
$9 : (in_6 : p_9, 1, \widetilde{p}_9),\ (out_6 : \widetilde{p}_9, 0, p_{10})$
$10 : (in_4 : p_{10}, 0, \widetilde{p}_{10}),\ (out_4 : \widetilde{p}_{10}, -1, p_0),\ (out_4 : \widetilde{p}_{10}, 0, p_{10})$
$11 : (in_5 : p_{11}, 0, \widetilde{p}_{11}),\ (out_5 : \widetilde{p}_{11}, -1, p_{12}),\ (out_5 : \widetilde{p}_{11}), 0, p_{13})$
$12 : (in_5 : p_{12}, 0, \widetilde{p}_{12}),\ (out_5 : \widetilde{p}_{12}, -1, p_{14}),\ (out_5 : \widetilde{p}_{12}), 0, p_{15})$
$13 : (in_2 : p_{13}, 0, \widetilde{p}_{13}),\ (out_2 : \widetilde{p}_{13}, -1, p_{18}),\ (out_2 : \widetilde{p}_{13}), 0, p_{19})$
$14 : (in_5 : p_{14}, 0, \widetilde{p}_{14}),\ (out_5 : \widetilde{p}_{14}, -1, p_{16}),\ (out_5 : \widetilde{p}_{14}), 0, p_{17})$
$15 : (in_3 : p_{13}, 0, \widetilde{p}_{13}),\ (out_3 : \widetilde{p}_{13}, -1, p_{18}),\ (out_3 : \widetilde{p}_{13}), 0, p_{20})$
$16 : (in_4 : p_{16}, 1, \widetilde{p}_{16}),\ (out_4 : \widetilde{p}_{16}, 0, p_{11})$
$17 : (in_2 : p_{17}, 1, \widetilde{p}_{17}),\ (out_2 : \widetilde{p}_{17}, 0, p_{21})$
$18 : (in_4 : p_{18}, 0, \widetilde{p}_{18}),\ (out_4 : \widetilde{p}_{18}, -1, p_0),\ (out_4 : \widetilde{p}_{18}), 0, p_{22})$
$19 : (in_0 : p_{19}, 0, \widetilde{p}_{19}),\ (out_0 : \widetilde{p}_{19}, -1, p_0),\ (out_3 : \widetilde{p}_{19}), 0, p_{18})$
$20 : (in_0 : p_{20}, 1, \widetilde{p}_{20}),\ (out_0 : \widetilde{p}_{20}, 0, p_0)$
$21 : (in_3 : p_{21}, 1, \widetilde{p}_{21}),\ (out_3 : \widetilde{p}_{21}, 0, p_{18})$

Fig. 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 1.

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 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.

Since UREM P systems are a relatively compact model of P systems, it is interesting to ask how many bits we would further save by compressing the above bit string. By operating like we did with register machines, 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(UREM), the theoretical description size complexity of the class of UREM P systems.

By assuming ds(DRM) = 263 and ds(UREM) = 921, an approximated value of the *redundancy* of UREM P systems with respect to deterministic register machines is:

$$R_{\text{DRM}}(\text{UREM}) = \frac{ds(\text{UREM})}{ds(\text{DRM})} = \frac{921}{263} \approx 3.50$$

On the other hand, by assuming ds(DRM) = 258 and ds(UREM) = 912 (that result by considering the entropies of the corresponding encoded strings) the redundancy becomes $R_{\text{DRM}}(\text{UREM}) = \frac{912}{258} \approx 3.53$. 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 and Directions for Further Research

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 four classes of computation devices (deterministic register machines, Turing 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. Further, we have seen that the smallest (standard) universal Turing machine currently known has about the same size of the smallest deterministic register machine. An intriguing 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 (including, for example, the small universal tissue and antiport P systems considered in [4,23,6])? In other words: What is the minimum number of bits which are necessary to describe the structure of a universal computing device?

Different 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 this point of view is a direction of research of a clear interest. Let us also note that we did not formally specify the encoding and decoding functions for our computational models. Finally, we did not count the size of these functions when calculating the description size of our computation devices. Whether this choice 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 of Milano–Bicocca.

References

1. Baiocchi, C.: Three small universal turing machines. In: Margenstern, M., Rogozhin, Y. (eds.) MCU 2001. LNCS, vol. 2055, pp. 1–10. Springer, Heidelberg (2001)
2. Balcázar, J.L., Díaz, J., Gabarró, J.: Structural Complexity, vol. I and II. Springer, Heidelberg (1988–1990)
3. Cook, M.: Universality in elementary cellular automata. Complex Systems 15, 1–40 (2004)
4. Csuhaj–Varjú, E., Margenstern, M., Vaszil, G., Verlan, S.: On small universal antiport P systems. Theoretical Computer Sci. 372, 152–164 (2007)
5. Freund, R., Leporati, A., Oswald, M., Zandron, C.: Sequential P systems with unit rules and energy assigned to membranes. In: Margenstern, M. (ed.) MCU 2004. LNCS, vol. 3354, pp. 200–210. Springer, Heidelberg (2005)
6. Freund, R., Oswald, M.: Small universal antiport P systems and universal multiset grammars. In: Proc. 4$^{\text{th}}$ Brainstorming Week on Membrane Computing, RGNC Report 03/2006, Fénix Editora, Sevilla, vol. II, pp. 51–64 (2006)
7. Ionescu, M., Păun, A., Păun, Gh., Jesús Pérez-Jímenez, M.: Computing with spiking neural P systems: Traces and small universal systems. In: Mao, C., Yokomori, T. (eds.) DNA12. LNCS, vol. 4287, pp. 1–16. Springer, Heidelberg (2006)
8. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae 71, 279–308 (2006)
9. Korec, I.: Small universal register machines. Theoretical Computer Sci. 168, 267–301 (1996)
10. Kudlek, M.: Small deterministic Turing machines. Theoretical Computer Sci. 168, 241–255 (1996)
11. Kudlek, M., Rogozhin, Y.: A universal turing machine with 3 states and 9 symbols. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 311–318. Springer, Heidelberg (2002)

12. Minsky, M.L.: Size and structure of universal Turing machines using Tag systems. In: Recursive Function Theory, Symp. in Pure Mathematics, pp. 229–238. American Mathematical Society, Providence (1962)
13. Neary, T., Woods, D.: Small fast universal Turing machines. Theoretical Computer Sci. 362, 171–195 (2006)
14. Neary, T., Woods, D.: Four small universal turing machines. In: Durand-Lose, J., Margenstern, M. (eds.) MCU 2007. LNCS, vol. 4664, pp. 242–254. Springer, Heidelberg (2007)
15. Woods, D., Neary, T.: Small semi-weakly universal turing machines. In: Durand-Lose, J., Margenstern, M. (eds.) MCU 2007. LNCS, vol. 4664, pp. 303–315. Springer, Heidelberg (2007)
16. Neary, T., Woods, D.: Small weakly universal Turing machines, arXiv [cs.CC]: 0707.4489v1 (2007)
17. Păun, A., Păun, Gh.: Small universal spiking neural P systems. BioSystems 90, 48–60 (2007)
18. Păun, Gh.: Computing with membranes. J. Computer and System Sci. 61, 108–143 (2000)
19. Păun, Gh.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
20. Rogozhin, Y.: Seven universal Turing machines. Math. Issled 69, 76–90 (1982)
21. Rogozhin, Y.: A universal Turing machine with 10 states and 3 symbols. Izv. Akad. Nauk. Respub. Moldova Mat. 4, 80–82, 95 (1992)
22. Rogozhin, Y.: Small universal Turing machines. Theoretical Computer Sci. 168, 215–240 (1996)
23. Rogozhin, Y., Verlan, S.: On the rule complexity of universal tissue P systems. In: Freund, R., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 356–362. Springer, Heidelberg (2006)
24. Shannon, C.E.: A universal Turing machine with two internal states. Automata Studies, Ann. Math. Stud. 34, 157–165 (1956)
25. Sipser, M.: Introduction to the Theory of Computation. PWS Publishing Co. (1997)
26. Wolfram, S.: A New Kind of Science. Wolfram Media, Champaign (2002)
27. 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

Abstract. 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.

Finite cumulative multisets are an extension of finite sets, such as $[a, a, b, c]$, or $[a, a, b], [a, a, b], [b]]$, where elements are put inside bracket pairs ("membranes"), and any element can occur more than one time. In a membrane structure all elements are built by starting from the empty multiset [ ] ("elementary membrane"). The most external pair of brackets is called "skin", and the elements inside the skin are called components of the structure.

The set $\mathbb{M}$ of finite membrane structures can be defined by induction by means of the multiset sum $+$ (summing the element occurrences of two multisets), which is a binary commutative and associative operation, and by means of the multiset singleton operation which, given a multiset S, provides the multiset $[S]$ having S as its unique element.

$$[\,] \in \mathbb{M} \qquad \textit{Base step}$$
$$S,\ S_1,\ S_2 \in \mathbb{M} \implies [S],\ S_1 + S_2 \in \mathbb{M} \qquad \textit{Inductive step}$$

For example,
$[[\,]] \in \mathbb{M}$ and $[[\,]] + [[\,]] = [[\,], [\,]] \in \mathbb{M}$.

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 292–298, 2009.

2 Proto-Agglomerates, Neo-Agglomerates and Conglomerates

Given n elements, the number of multisets of k elements over the n elements, according to [1], is given by:

$$\binom{n+k-1}{k} \tag{1}$$

By using formula (1), the following recurrent formula is given in Knuth's book [4] (pag. 386), which provides the number $T(n)$ unlabeled unordered trees of n nodes (membrane structures with n membranes), where $\mathbb{N}$ is the set of natural numbers, $n > 0 \in \mathbb{N}$, $T(0) = 1$, and $j, n_1, n_2, \ldots, n_j, k_1, k_2, \ldots, k_j \in \mathbb{N}$:

$$T(n+1) = \sum_{k_1 \cdot n_1 + k_2 \cdot n_2, \ldots k_j \cdot n_j = n} \prod_{i=1,\ldots,\, j} \binom{T(n_i) + k_i - 1}{k_i} \tag{2}$$

Unfortunately, formula (2) is not manageable for an effective computation, because it is based on integer partitions, which grow, according to Ramanujan's exponential

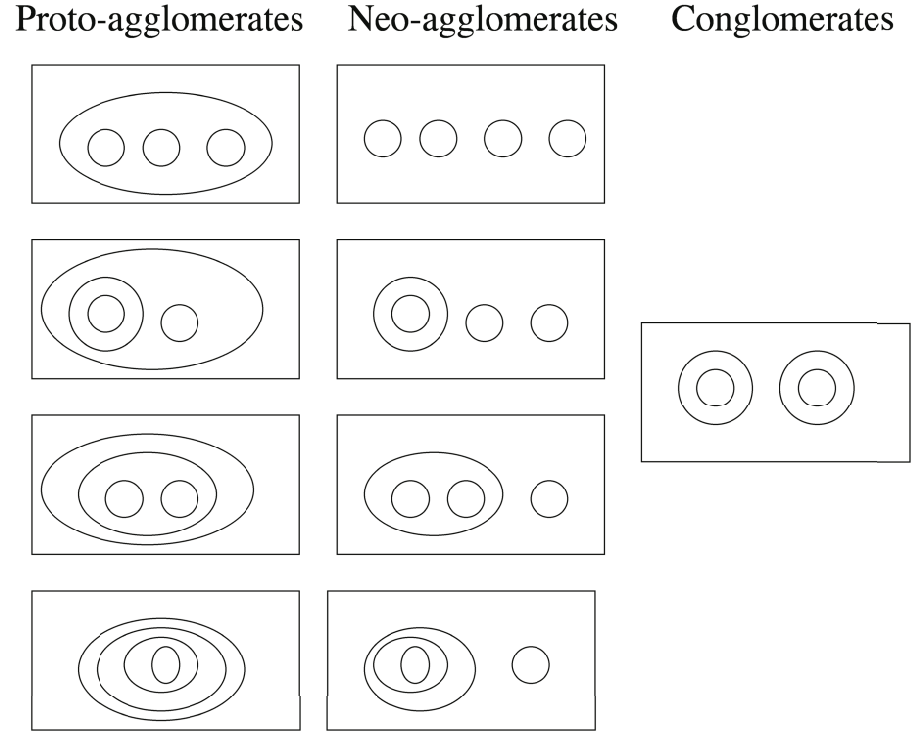


Fig. 1. A representation of the membrane structures with four membranes (the skin is represented by a rectangle)

asympthotic formula [3], making the evaluation of the sum in formula (2) very complex. Now, we adopt an alternative enumeration strategy, by counting all the possible different membrane structures without considering the skin. For this reason we make the following partition of membrane structures, based on the structure of membranes inside the skin: i) **Proto-agglomerates**, ii) **Neo-agglomerates**, and iii) **Conglomerates**. Proto-agglomerates are structure where the skin contains only a singleton multiset. Neo-agglomerates are structure where the skin contains empty multisets. Conglomerates are structures different from proto-agglomerates and neo-agglomerates, that is, inside the skin there is not only a singleton and there are not empty multisets. In Fig. 1 structures on the left side are proto-agglomerates, structures in the middle are neo-agglomerates, and the structure on the right is a conglomerate (the skin is represented by a rectangle). Proto-agglomerates are essentially rooted unlabeled unordered trees, while conglomerates and neo-agglomerates represent unlabeled unordered forests. We denote by $M(n) = T(n+1)$ the number of membrane structures having n membranes (pairs of matching brackets) inside the skin, and by $P(n), N(n), C(n)$, the number of proto-agglomerates, neo-agglomerates, and conglomerates, respectively, having n membranes inside the skin (the skin will not mentioned anymore and it is not counted in the number of membranes). It easy to realize that a membrane structure with n membranes, when it is put inside a further membrane, provides a proto-agglomerate with $n+1$ membranes, while united with the multiset $[[\ ]]$ provides a neo-conglomerate with $n+1$ membranes. 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 $n > 0$, $M(n+1) = 2M(n) + C(n+1)$.*

Lemma 3. $C(1) = C(2) = C(3) = 0$. *For $n > 0$, $C(n+1) \leq M(n)$.*

Proof. Removing the external membrane in a component of a conglomerate with $n+1$ membranes, provides a membrane structure with n membranes. Therefore conglomerates with $n+1$ membranes are at most $M(n)$.

Lemma 4. *Let $C_i(n)$ denote the number of conglomerates having n membranes and exactly i components, then for $n > 2$*

$$C(n) = \sum_{i=2,\lfloor n/2 \rfloor} C_i(n).$$

Proof. At most $\lfloor n/2 \rfloor$ components can be present in a conglomerate with n membranes. Putting together lemmas 2 and 3 we get the following lemma.

Lemma 5. *For $n > 1$*

$$2M(n) \leq M(n+1) \leq 3M(n).$$

$$2^n \leq M(n+1) \leq 3^n.$$

According to the previous lemmas, we see that in the number $M(n+1)$ the part $2M(n)$ refers to proto-agglomerates plus neo-agglomerate. 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 conglomerates 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 (skin brackets are not indicated).

1	[ ]
2	[ ], [ ] [[ ]]
3	[ ], [ ], [ ] [[[ ]]] [ ], [[ ]] [[ ], [ ]]

From lemma 6 we evaluate immediately $M(4) = 2M(3)+1 = 9$. In fact, $C(4) = 1$, because there i s only a conglomerate with 4 membranes: [[ ]], [[ ]]. Analogously, $M(5) = 2M(4) + 2 = 18 + 2 = 20$, because there are two conglomerates 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 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. 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 $X \in \mathbb{N}^*$ of length n.

$$\bigotimes^i X = \sum_{\mu \in \Pi_{\mathbf{n-i+1},\,\mathbf{i}}} \prod_{j \in \mu} \binom{X(j) + \mu(j) - 1}{\mu(j)} \tag{3}$$

For $i, j \in \mathbb{N}$, let $M(1, .., j)$ denote the sequence $(M(1), \ldots, M(j))$, then the main lemma follows.

Lemma 6. *For $n > 2$*

$$C(n+1) = \sum_{i=2,\lfloor (n+1)/2 \rfloor} \bigotimes^i M(1, .., n). \tag{4}$$

Proof Outline. Conglomerates 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 wrapping these i components,

therefore the remaining $n+1-i$ are partitioned among these components in all the possible ways. In conglomerates with 2 components $n+1-2$ membranes can be distributed in 2 components. In conglomerates with 3 components $n+1-3$ membranes can be distributed in 3 components, and so on, up to $n+1-\lfloor (n+1)/2 \rfloor$ membranes in $\lfloor (n+1)/2 \rfloor$ components. In order to compute the number of all possible membrane arrangements, each partition 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. For example, if a partition has three parts, with two equal parts, say $n+1-3 = p+p+q$, then in a corresponding conglomerate 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. 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, 4, and 6 the final proposition follows. The application of the formula of lemma 6, 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 6 the value $C(11)$ is given by:

$$C(11) = \sum_{i=2,5} \bigotimes^i M(1,\ldots,10)$$

Now, according to formula (3), we need to compute the right member of this equation for the values $\Pi_{9,2}, \Pi_{8,3}, \Pi_{7,4}, \Pi_{6,5}$ corresponding to the values $2,3,4,5$ of i.

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:

$$\mathit{\Pi}_{8,3} = \{\{6,1,1\},\{5,2,1\},\{4,3,1\},\{4,2,2\},\{3,3,2\}\}$$

therefore:

$$\bigotimes^3 M(1,\ldots,10) = \sum_{\mu\in\mathit{\Pi}_{8,3}} \prod_{j\in\mu} \binom{M(j)+\mu(j)-1}{\mu(j)}$$

$$\left[\binom{48+1-1}{1}\binom{1+2-1}{2}\right] + \left[\binom{20+1-1}{1}\binom{2+1-1}{1}\binom{1+1-1}{1}\right] + \left[\binom{9+1-1}{1}\binom{4+1-1}{1}\binom{1+1-1}{1}\right] + \left[\binom{9+1-1}{1}\binom{2+2-1}{2}\right] + \left[\binom{4+2-1}{2}\binom{2+1-1}{1}\right] = 48 + 20\cdot 2 + 9\cdot 4 + 9\cdot 3 + 10\cdot 2 = 171.$$

The integer partitions of 7 in four summands yield the following set:

$$\mathit{\Pi}_{7,4} = \{\{4,1,1,1\},\{3,2,1,1\},\{2,2,2,1\}\}$$

therefore:

$$\bigotimes^4 M(1,\ldots,10) = \sum_{\mu\in\mathit{\Pi}_{7,4}} \prod_{j\in\mu} \binom{M(j)+\mu(j)-1}{\mu(j)} =$$

$$\left[\binom{9+1-1}{1}\binom{1+3-1}{3}\right] + \left[\binom{4+1-1}{1}\binom{2+1-1}{1}\binom{1+2-1}{2}\right] + \left[\binom{2+3-1}{3}\binom{1+1-1}{1}\right] =$$

$$9 + 4\cdot 2 + 4 = 21.$$

The integer partitions of 6 in five summands yield the following set:

$$\mathit{\Pi}_{6,5} = \{\{2,1,1,1,1\}\}$$

therefore:

$$\bigotimes^5 M(1,\ldots,10) = \sum_{\mu\in\mathit{\Pi}_{6,5}} \prod_{j\in\mu} \binom{M(j)+\mu(j)-1}{\mu(j)} =$$

$$\left[\binom{2+1-1}{1}\binom{1+4-1}{4}\right] = 2.$$

In conclusion, $C(11) = 888 + 171 + 21 + 2 = 1082$, therefore:

$$M(11) = 2M(10) + 1082 = 4766.$$

References

1. Aigner, M.: Discrete Mathematics. American Mathematical Society (2007)
2. Cayley, A.: On the analytical forms called trees, with application to the theory of chemical combinations. Mathematical Papers 9, 427–460 (1875)
3. Conway, J.H., Guy, R.K.: The Book of Numbers. Springer, Heidelberg (1996)
4. Knuth, D.: The Art of Computer Programming. Fundamental Algorithms, vol. 1. Addison Wesley, Reading (1968)

5. Otter, R.: The number of trees. The Annals of Mathematics 2nd Ser., 49, 583–599 (1948)
6. Păun, G.: Personal Communication (October 1998)
7. Păun, G.: Computing with membranes. J. Comput. System Sci. 61, 108–143 (2000)
8. Sloane, N.J.A.: The on-line encyclopedia of integer sequences. Notices of The American Mathematical Society 59, 912–915 (2003)

Toward an MP Model of Non-Photochemical Quenching

Vincenzo Manca[1], Roberto Pagliarini[1], and 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

Abstract. In this paper we apply the formalism of metabolic P systems for modeling an important phenomenon of photochemical organisms, which determines the plants 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 behavior 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 prevents ROS formation, is called *non photochemical quenching*, shortly *NPQ*. Through this phenomenon the excess of light can, in fact, be dissipated by using 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 [20] and developed in [15] 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 [22] for a better understanding of quantitative aspects of biological systems, meanwhile avoiding the use of complex systems of

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 299–310, 2009.

differential equations. Differently from the classical P systems [8,26,27], based on nondeterministic evolution strategies, MP systems have discrete evolutions computed by deterministic algorithms called *metabolic algorithms* [20], based on a new perspective introduced in [22] and then developed in a series of papers [5,6,16,18,21,17,19,11]. 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 [11] it has been demonstrated that exists, under suitable hypotheses and with some approximation, an equivalence between ODE systems and MP models, while in [7] it was show the equivalence of a class of Petri nets with MP systems..

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 [5,22], the suitable-infect-recovered epidemic [5], the protein kinase C activation [6], the circadian rhythms [10] and the mitotic cycles in early amphibian embryos [21].

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 [24,25].

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, like ATP and NADPH, involved in the so-called *dark phase* of photosynthesis (fixation of atmospheric CO_2 to carbohydrates [3], Figure 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 1, arrow 2), they can emit fluorescence radiation (Figure 1, arrow 3), or can decade in the *triplet state*, that can be transferred to oxygen atoms thus generating ROS (Figure 1, arrow 9).

In our 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 photosystem is able to accept photons).

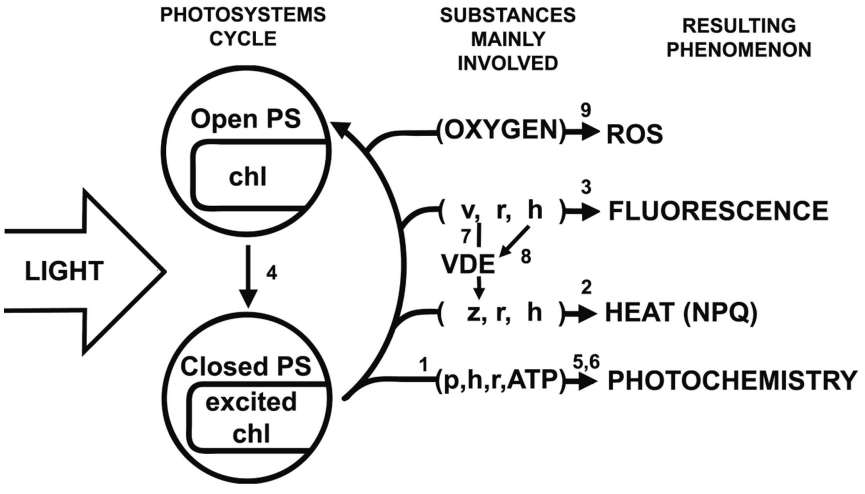


Fig. 1. Main actors and relationships of NPQ phenomenon of excitation/deexcitation chlorophylls diagram, according to the following abbreviations: chl = chlorophyll, ps = photosytem, h = hydrogen ion, r = reactivity, p = NADPH, VDE = violaxanthin de-epoxidase, v = violaxanthin, z = zeaxanthin

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 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 specific functions, associated with reactions, called *flux regulations maps* or *flux maps*. The notion of MP system we use comes essentially from [15], 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 $\alpha_r \rightarrow \beta_r$, with α_r, β_r strings over the alphabet X. In particular, 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;*
- $V = \{v_1, v_2, \ldots, v_k\}$, $k \in \mathbb{N}$, *is a finite set of parameters provided by a set $H = \{h_v | v \in V\}$ of parameters evolution functions. The function $h_v : \mathbb{N} \rightarrow \mathbb{R}$ 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 $q : X \cup V \rightarrow \mathbb{R}$. If we consider 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 $\varphi_r : Q \rightarrow \mathbb{R}$ 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, $\sigma_0 = (X[0], V[0])$;*
- $\delta : \mathbb{N} \rightarrow Q$ *is the dynamics 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:*

$$X[i+1] = A \times U[i] + X[i] \tag{1}$$

  where A is the stoichiometric matrix of R over X of dimension $n \times m$ while $\times$, $+$, are the usual matrix product and vector sum.

The key notion of the definition above is the set Φ of flux regulation maps. Their knowledge provide us the flux vectors, and consequently, the dynamics of an MP system according to equation (1). As we will show later on, flux regulation functions can be deduced from the macroscopic observations of system behaviors,

along a series of steps. Now, a fundamental question is about the meaning of these functions. Do they represent the real interactions between (bio)-molecules? Or are they only abstract mathematical constructs allowing us to model the observed phenomena?

The second case is surely main appropriate. For this reason, flux regulation maps emphasize the limitations of any macroscopic observational approach, but, at the same time, its advantages. In fact, we cannot pretend to know what exactly happen at a real microscopic level, but we can describe internal dynamical logics of processes which are located at a more abstract level with respect to the biomolecular one.

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 dynamics in accordance with the experiment, within an acceptable approximation, but which could also predict the future behavior of the process?"* We will define an MP model that answers 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 ability of the NPQ phenomenon. 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[1] (a.u.) from measurements on the sample [23]. NADPH produced has been estimated through laboratory measures, the pH value was deduced by combining data from literature [9,14,28] 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 [12]. The change over time of violaxanthin and zeaxanthin was obtained with lab measurements and the VDE activity [13].

[1] 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.

The NPQ process discussed in Section 2 can be expressed by the set of reactions given in Table 1. The first three reactions model the possible fates of excited chlorophylls (arrows 1, 2 and 3 in Figure 1), the fourth models the fact that the opened photosystems turn into closed photosystems (arrow 4 in Figure 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 1). Finally, the seventh and eighth represent xanthophylls cycle (arrows 7 and 8 in Figure 1).

In Table 1 are also indicated the set of *Tuners* related to each reaction. Tuners of a reaction r_i are quantities which influence, with their variations, the variations of the flux units.

Table 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

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

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^+:

$$f^+[j] = \sum_{i=0}^{j} f[i], \quad q^+[j] = \sum_{i=0}^{j} q[i] \tag{2}$$

where the values $f[i]$ and $q[i]$ represent, respectively, the amount of fluorescence and NPQ observed in the phenomenon at the step i.

Almost all elements occurring in the definition of MP system are known, 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 [15]. This means that the key point, for defining an MP system modeling 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 is 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 [20], 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 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 [29]. We use a discrete notion of log-gain, given by the following equation:

$$Lg(w[i]) = (w[i+1] - w[i])/w[i] \tag{3}$$

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:*

$$(U[i+1] - U[i])/U[i] = B \times L[i] \quad + \quad P[i+1] \tag{4}$$

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 ;*
- $P[i] = (p_r[i] |\ r \in R, p_r[i] = 0 \quad if \quad r \notin R_0)$ *is the column offset vector, where $R_0 \subseteq R$ is a set of reactions which has Covering Offset Log Gain Property [15]. This vector is constituted by the difference between tuner log-gains and flux log-gains;*
- $\times$ *denotes the usual matrix product;*
- $+, -, /$ *denote, when applied to vectors, the componentwise sum, subtraction, and division.*

We call $COLG[i]$ the system of equations obtained by (4).

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 $P[i+1]$:

$$\begin{bmatrix} 0&1&1&1&0&0&0&0&0&0&0&1&0&0&0 \\ 0&1&1&0&0&0&0&1&0&0&1&1&0&0&0 \\ 0&1&1&0&0&0&1&0&0&0&1&1&0&0&0 \\ 1&0&0&0&0&0&0&0&0&0&0&1&0&0&1 \\ 0&0&1&0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&1&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&1&0&0&0&0&0&0&0&0&0 \end{bmatrix}, \quad \begin{bmatrix} p_1[i+1] \\ p_2[i+1] \\ p_3[i+1] \\ 0 \\ p_4[i+1] \\ p_5[i+1] \\ p_6[i+1] \\ p_7[i+1] \end{bmatrix}$$

The transposed $L[i]^T$ of vector $L[i]$ of equation (4) 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 $SD[i]$ [20], which represents the substances variations

$$X[i+1] - X[i] = A \times U[i] \tag{5}$$

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 (4) and (5) 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 [15] 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 the NPQ process, where variables are in bold font and constitute the flux unit vector $U[i+1]$ and the offset vector $P[i+1]$:

$$\begin{aligned} A \times \mathbf{U[i+1]} &= X[i+2] - X[i+1] \\ (\mathbf{U[i+1]} - U[i])/U[i] &= B \times L[i] \quad + \quad \mathbf{P[i+1]} \end{aligned} \tag{6}$$

Now, as the vectors $X[i]$ and $V[i]$, for $1 \leq i \leq 810$, are given by experimental measures, we solve the system (6) 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 $COLG[i] + SD[i]$ and we used a suitable iterative technique [20] in order to solve a non-linear system of equations giving a good approximation of $U[0]$.

Given the values of the flux units 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 2, approximate the regulation function Φ associated to each reaction $r \in R$.

Table 2. NPQ reactions and flux regulation maps. The values of polynomial coefficients can be downloaded from [30].

Reactions	Flux regulation maps
$r_1 : c \rightarrow o + 12h + p$	$\varphi_{r_1} = \alpha_1 + \beta_1 ol + \gamma_1 cl + \eta_1 rl + \vartheta_1 hlp + \rho_1 \frac{v}{z} l$
$r_2 : c \rightarrow c + q^+$	$\varphi_{r_2} = \alpha_2 + \beta_2 c + \gamma_2 r + \eta_2 z + \vartheta_2 l + \rho_2 h$
$r_3 : c \rightarrow c + f^+$	$\varphi_{r_3} = \alpha_3 + \beta_3 ch + \gamma_3 v + \eta_3 r^{-1} l$
$r_4 : o \rightarrow c$	$\varphi_{r_4} = \alpha_4 + \beta_4 ol + \gamma_4 cl + \eta_4 rl + \vartheta_4 hlp + \rho_4 \frac{v}{z} l$
$r_5 : h \rightarrow \lambda$	$\varphi_{r_5} = \alpha_5 + \beta_5 ol + \gamma_5 cl + \eta_5 rl + \vartheta_5 hlp + \rho_5 \frac{v}{z} l$
$r_6 : p \rightarrow \lambda$	$\varphi_{r_6} = \alpha_6 + \beta_6 ol + \gamma_6 cl + \eta_6 rl + \vartheta_6 hlp + \rho_6 \frac{v}{z} l$
$r_7 : x + 100v \rightarrow x + 100z$	$\varphi_{r_7} = \alpha_7 + \beta_7 v + \gamma_7 x$
$r_8 : y + h \rightarrow x$	$\varphi_{r_8} = \alpha_8 + \beta_8 y + \gamma_8 h$

It is possible to see in Figure 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).

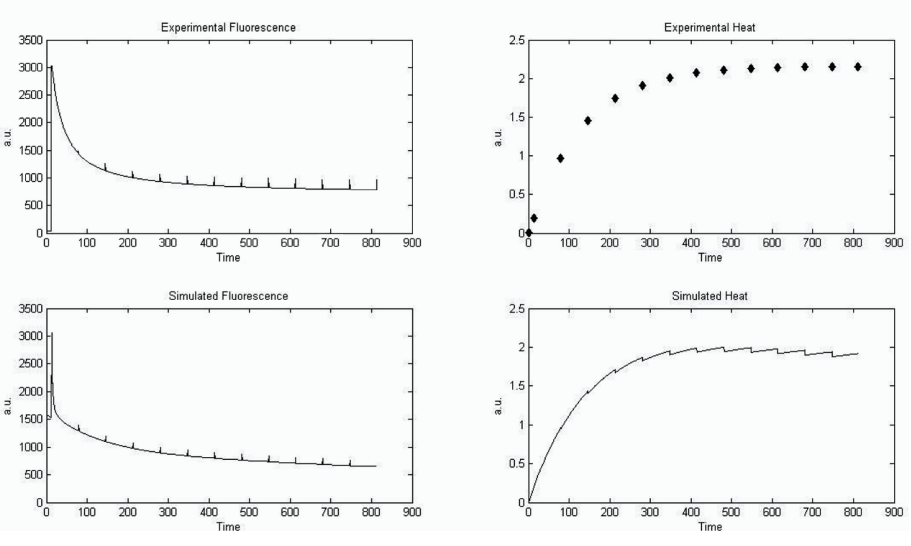


Fig. 2. Values of the fluorescence and NPQ, in a. u., during the experimental measurements (top) and simulation results obtained by MP system (bottom). Notice that while experimental heat is measured in correspondence of saturating flash, simulated one is continuous. Our results allow to hypothesize that also the NPQ values are perturbated by flash of light.

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 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 investigations 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.

Acknowledgement

We are grateful to Prof. Bassi's group of Biotechnological Department, at University of Verona, for laboratory analysis and Dott. Petronia Carillo, Department of Life Sciences, Second University of Naples, for CO_2 uptake measurements in relation to the experiment relevant to us.

References

1. Ahn, T.K., Avenson, T.J., Ballottari, M., Cheng, Y.C., Niyogi, K.K., Bassi, R., Fleming, G.R.: Architecture of a charge-transfer state regulating light harvesting in a plant antenna protein. Science 320(5877), 794–797 (2008)
2. Alder, N.N., Theg, S.M.: Energetics of protein transport across biological membranes: a study of the thylakoid ΔpH-dependent/cptat pathway. Cell 112, 231–242 (2003)
3. Benson, A., Calvin, M.: Carbon dioxide fixation by green plants. Annual Review of Plant Physiology and Plant Molecular Biology 1, 25–42 (1950)
4. von Bertalanffy, L.: General Systems Theory: Foundations, Developments, Applications. George Braziller Inc., New York (1967)

5. Bianco, L., Fontana, F., Franco, G., Manca, V.: P systems for biological dynamics. In: [8], pp. 81–126
6. Bianco, L., Fontana, G., Manca, V.: P systems with reaction maps. Intern. J. Foundations of Computer Sci. 17, 27–48 (2006)
7. Castellini, A., Franco, G., Manca, V.: Toward a representation of Hybrid Functional Petri Nets by MP systems. In: Proc. 2nd International Workshop on Natural Computing, IWNC 2007, Nagoya University, Japan. Springer, Heidelberg (2007)
8. Ciobanu, G., Păun, G., Pérez-Jiménez, M.J. (eds.): Applications of Membrane Computing. Springer, Heidelberg (2006)
9. Evron, Y., McCarty, R.E.: Simultaneous measurement of deltapH and electron transport in chloroplast thylakoids by 9-aminoacridine fluorescence. Plant Physiol. 124, 407–414 (2000)
10. Fontana, F., Bianco, L., Manca, V.: P systems and the modeling of biochemical oscillations. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 199–208. Springer, Heidelberg (2006)
11. Fontana, F., Manca, V.: Discrete solution to differential equations by metabolic P systems. Theoretical Computer Sci. 372, 165–182 (2007)
12. Gisselsson, A., Szilagyi, A., Akerlund, H.: 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)
13. Holzwarth, A.R.: Applications of ultrafast laser spectroscopy for the study of biological systems. Q. Rev. Biophys. 22, 239–295 (1989)
14. Kanazawa, A., Kramer, D.M.: In vivo modulation of nonphotochemical exciton quenching (NPQ) by regulation of the chloroplast atp synthase. PNAS 99, 12789–12794 (2002)
15. Manca, V.: Log-gain principles for metabolic P systems (submitted, 2008)
16. Manca, V.: Topics and problems in metabolic P systems. In: Proc. Fourth Braintorming Week on Membrane Computing, Fenix Editora, Sevilla (2006)
17. Manca, V.: Metabolic P systems for biochemical dynamics. Progress in Natural Science 17, 384–391 (2007)
18. Manca, V.: MP systems approaches to biochemical dynamics: Biological rhythms and oscillations. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 86–99. Springer, Heidelberg (2006)
19. Manca, V.: Discrete simulations of biochemical dynamics. In: Garzon, M.H., Yan, H. (eds.) DNA 2007. LNCS, vol. 4848, pp. 231–235. Springer, Heidelberg (2008)
20. Manca, V.: The metabolic algorithm for P systems: Principles and applications. Theoretical Computer Sci. 404, 142–157 (2008)
21. Manca, V., Bianco, L.: Biological networks in metabolic P systems. BioSystems 91, 489–498 (2008)
22. Manca, V., Bianco, L., Fontana, F.: Evolution and oscillation in P systems: Applications to biological phenomena. In: Mauri, G., et al. (eds.) WMC 2004. LNCS, vol. 3365, pp. 63–84. Springer, Heidelberg (2005)
23. Maxwell, K., Johnson, G.N.: Chlorophyll fluorescence - a practical guide. Journal of Experimental Botany 51, 659–668 (2000)
24. Nelson, N., Ben-Shem, A.: The complex architecture of oxygenic photosynthesis. Nature Reviews Molecular Cell Biology 5, 971–982 (2006)
25. Nelson, N., Yocum, C.: Structure and function of photosystems I and II. The Annual Review of Plant Biology 57, 521–565 (2006)
26. Păun, G.: Computing with membranes. J. Computer and System Sci. 61, 108–143 (2000)

27. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
28. Trubitsin, B.V., Tikhonov, A.N.: Determination of a transmembrane pH difference in chloroplasts with a spin label tempamine. Journal of Magnetic Resonance 163, 257–269 (2003)
29. Voit, E.O.: Computational Analysis of Biochemical Systems. Cambridge University Press, Cambridge (2000)
30. 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

Mathematical Institute, Leiden University
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
muskulus@math.leidenuniv.nl

Abstract. The page rank of a webpage is a numerical estimate of its authority. 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 render the transition matrix irreducible and aperiodic. The same idea can be applied to P systems, and the resulting invariant probability distribution characterizes their dynamical behavior, analogous to recurrent states in deterministic dynamical systems. The modification made to the original P system gives rise to a new class of P systems with the property that their computations need to be robust against random mutations. Another application is the pathway identification problem, where a metabolite graph is constructed from information about biochemical reactions available in public databases. The invariant distribution of this graph, properly interpreted as a Markov chain, should allow to search pathways more efficiently than current algorithms. Such automatic pathway calculations can be used to derive appropriate P system models of metabolic processes.

1 Introduction and Background

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 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 one 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

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 311–324, 2009.

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[1], but Google states that the "heart of their software" is an algorithm called *PageRank* [16], whose name seems to be inspired by the last name of Google founder Lawrence Page [38].

The original ranking algorithm behind Google has been published [8,33] 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* [2]. 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 [28].

In the following we will describe applications of page ranking in the area of membrane systems [34,35]. We will specifically concentrate on the original *PageRank* algorithm, since it is closely related to Markov chain modeling of dynamical P systems as in [31]. The applications that we will discuss are (i) defining the *recurrent behavior* of dynamical P systems, which results in (ii) a new *complexity measure* for dynamical P systems; (iii) proposing a new class of P systems with interesting robustness properties, and (iv) discussing applications in the *identification* of P systems, where biochemical databases are used to infer 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 authoritative page u itself is, the higher its vote for page v should count, leading to a recursive definition as follows. Let

$$\begin{aligned} r: \quad & V \to \mathbb{R}_+ \\ & v \mapsto r(v) \end{aligned}$$

[1] 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

$$r(v) := \sum_{u \in \{V \mid (u,v) \in A\}} \frac{r(u)}{\text{outdeg}(u)}$$

where the sum runs over all nodes u linking to the page v and $\text{outdeg}(u)$ is the out-degree 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 [7]. Since the webgraph is finite, the function r can be normalized such that $\sum_{v \in V} r(v) = 1$. One can then interpret $r \in \mathbb{R}_+^{|V|}$ as a probability distribution over the set V of webpages. The *transition matrix*

$$P_{uv} = \begin{cases} 1/\text{outdeg}(u) & \text{if } (u,v) \in A, \\ 0 & \text{if } (u,v) \notin A \end{cases}$$

then corresponds to the model for a person surfing between web pages, from now on simply addressed as a *surfer*, as described in [8]. 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 the probability distribution r, 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 nodes 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 [20]. 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 probability)? 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, 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

$$\bar{\bar{P}}_{uv} = \begin{cases} 1/|V| & \text{if } \mathrm{outdeg}(u) = 0, \\ \alpha/|V| & \text{if } \mathrm{outdeg}(u) > 0 \text{ and } (u,v) \notin A, \\ \alpha/|V| + (1-\alpha)/\mathrm{outdeg}(u) & \text{if } (u,v) \in A \end{cases} \tag{1}$$

which is truly an irreducible and aperiodic matrix [26]. 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:

$$\lim_{n\to\infty} (P^t)^n x_0 = r, \tag{2}$$

independent of the initial distribution x_0, i.e., his/her starting page. 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 [5], 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) transition matrix of the webgraph is very difficult, due to the graph's enormous size. On the other hand, Eq. 2, starting from the uniform distribution $x_0(u) = 1/|V|$ is used in practice, and is usually called the *power method* [15]. See [20] for further improvements.

3 P Systems and the Random Surfer Model

P system is a general term to describe a broad class of unconventional models of computation that are usually based on multiset rewriting in a hierarchical structure of so-called membranes [35], but also include computational models based on other mechanisms, for example string or grammar rewriting. Originally introduced by Gheorghe Păun in a seminal paper [34], nowadays there exists a large community of researchers working on and with different extensions and variants of P systems.

How are we to interpret the above changes in the context of P systems, i.e., when we are thinking about the random surfer model with possible jumps (Eq. 1) not only as mathematically sufficient and convenient, but rather as a 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. Before discussing the problems and possible solutions, the following example illustrates the notion of an invariant distribution in a simple class of P systems.

Let us consider the case of a *probabilistic P system* as in [11]. Starting from an initial configuration (multiset) c_0 the evolution of a probabilistic P system generates a rooted tree S of possible states, where each state $i \in S$ is encountered with a probability $p_{c_0,i}$ during the computation (confer [35]). The leaves $\mathcal{L} \subset S$ of this tree are the halting states and each halting state $h \in \mathcal{L}$ is reached with a 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, the state space has the structure of an irreducible Markov chain. This Markov chain could be periodic, but it is easy to see that for a such a finite "closed tree" an invariant probability distribution exists as in the case of an irreducible and aperiodic Markov chain. In fact, the unique invariant distribution is given by $\mu_i = 1/|S| \cdot p_{c_0,i}$ for each state $i \in S$. The factor $1/|S|$ has been introduced such that $\sum_{i\in S} \mu_i = 1$.

We see that the concept of invariant distribution generalizes the probability of reaching a halting state in a probabilistic P system. However, the requirement that the evolution has a tree structure makes the class of probabilistic P systems very special. Moreover, the evolution of such a system is not the same as the dynamics described by Eq. 1.

We consider *flat* P systems in the following, i.e., P systems with exactly one membrane. It is well known that each static[2]P system with k membranes is isomorphic to a flat P system, so this is no restriction.

Problem 1. The state space of P systems is usually not known a priori.

Remark 1. Generating the state space of a P system corresponds to the well known *reachability problem*. But if the P system operates in the *asymptotic regime*, i.e., when there are enough objects in the system such that all rules are

[2] 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.

applicable, it can be considered a vector-addition system on the infinite lattice $\mathbb{Z}^n$, where n is the number of distinct objects. The state space is the affine image of $\mathbb{R}^m$ (m being the number of rules) under the stoichiometric map M, which for a given initial condition $c \in \mathbb{Z}^n$ is the sublattice $c + M\mathbb{R}^m$ of $\mathbb{Z}^n$. In this case, the geometry of the state space is easy to understand and reachability can be efficiently tested [31].

In general P systems, we are often only interested in a finite subset $Q \subset S$ of state space S, and the restriction of the invariant distribution to Q. Due to Eq. 2 the invariant distribution on Q can be approximated by *simulating* the P system a large number of times, *provided* that it is aperiodic and irreducible.

Problem 2. The state space of P systems is usually infinite.

Remark 2. This is a variant of the previous problem. In principle, for a countably infinite state space there can still exist invariant distributions (see [7] for results about when this is known to be the case). However, some problematic issues surface with an infinite state space for the teleportation property. In particular, the probability of teleportating from state $i \in V$ to some state $j \in V$ is equal to zero[3] when $|V| = \infty$. The only solution of this problem is to somehow modify the teleportation property (confer Section 4).

Problem 3. P systems are non-deterministic, and not probabilistic. What sense does an invariant probability distribution make for a non-deterministic system?

Remark 3. The easy solution of this problem is to only consider variants of P systems that are probabilistic instead of being non-deterministic (see Section 5).

However, for the sake of the argument, let us consider a truly non-deterministic system. It can be considered as the equivalence class of all probabilistic systems with non-zero transition probabilities $P_{ij} > 0$ exactly for all states $i, j \in V$, where j is reachable from i in one time step. An invariant distribution has the property that its support is the whole state space, so the notion of invariant distribution for a non-deterministic system is equivalent with the information what the state space is. Note that quantitative information *can* be obtained in non-deterministic systems (see the next remark for an example).

Problem 4. If we consider a subset $Q \subset S$ of the state space S of a P system, can we find the invariant distribution restricted to Q?

Remark 4. This is probably *the most important* problem from a practical point of view. As already discussed in the first remark, when the P system is aperiodic and irreducible, the invariant distribution can be approximated by simulation. However, when the system leaves the subset Q during the simulations, knowledge about the dynamics outside of Q is needed, so this is not completely satisfying.

[3] Mathematically, although there does not exist a uniform probability distribution on V then, it is still possible to jump to a *random* element $j \in V$ with probability $\alpha > 0$ at each time step, when a probabilistic version of the *axiom of choice* is assumed. However, this will lead too far here, as it is not of practical importance.

The main problem with the calculation of the invariant distribution on Q is that the flow of probability from $S \setminus Q$ into Q is not known. However, for the invariant distribution the flow is in *equilibrium*, i.e., the total outflow from Q into its complement equals the total of the unknown inflows. By looping back each outflow into the inflows it is possible to constrain the possible invariant distributions[4] of Q, but it seems unlikely that they can be uniquely identified.

When inflows and matching outflows are *prescribed*, however, it is always possible to determine a corresponding invariant distribution on Q.

Problem 5. When is this invariant distribution compatible with prescribed outflows?

Remark 5. The answer is very simple: it never is, generically. So now it is important to find algorithmic ways of adjusting (and thereby violating) the inflow conditions such that the total sum of inflow and outflow violations is minimized.

Another related and very useful quantity, which *can* be computed by local information only is the mean *escape time*[5] from a subset $Q \subset S$ of state space.

4 Aperiodic and Irreducible P Systems

When a P system with probabilistic state transitions has an aperiodic and irreducible transition matrix, a unique invariant probability distribution exists. However, Google's random surfer model is not the only way to achieve these properties of the transition matrix.

For example, when all rules in a P system are irreversible, the system is irreducible. Unfortunately, in such a system the question of periodicity is unclear.

Let us now consider a more general situation. Assume that at each time step[6] there is a small probability for each object to change spontaneously into another object, analogous to mutations in DNA. The objects undergoing such a change

[4] Let Q be a finite set of order k. Consider the modified $(k+1)$-by-$(k+1)$ transition matrix Q_k that describes the transitions inside Q and from Q to an external state x (which represents all states outside of Q) and back from x into the k-th state of Q (with probability 1). Denote the unique invariant distribution of this matrix by r_i. Due to the linearity at the level of distributions, the true invariant distribution r, restricted to Q, is a linear combination of the first k components of all r_i. Of course, this only holds when $\alpha = 0$, but the result can easily be generalized to $\alpha > 0$ also.

[5] There are subtle connections between (i) this quantity, (ii) the page ranks of the states in Q, and (iii) the number of loops in Q. Basically, there are two contributions to the invariant distribution r on Q: Part of r consists of probability that flows into Q from outside of Q, and another part of r results from probability that flows around inside of Q in loops. However, this connection is not well understood at the moment, and further research is required.

[6] 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.

cannot be used in another rule at this time step. By adding rules of the form $u \to v$ for each possible pair of objects (u, v), we can realize this. Let us call a P system with this property a *leaky* P system. Note that leaky P systems are not always irreducible, as the 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.

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? What kind of error-correcting (repair) mechanisms can be envisaged in P systems? Moreover, the following two theoretical problems exist:

Problem 6. Given an irreducible aperiodic Markov chain, when adding the teleportation property of Eq. 1, do the corresponding invariant distributions $\mu(\alpha)$ converge in the limit $\alpha \to 0$?

Remark 6. Numerical studies have been done in case of the webgraph (confer [26] for references).

Problem 7. Although the random surfer model does not apply to a leaky P system, does the invariant distribution of a leaky P system converge against the same invariant distribution as in the random surfer model (of the same underlying P system), in the limit that $\alpha \to 0$?

Remark 7. This is the case if the leaky P system has the same communicating classes as the corresponding random surfer system P.

5 Recurrent Behavior and Complexity

A particular interesting application area for P systems is the emerging discipline of systems biology [24], and in the past years a number of biological systems have been simulated and analyzed by P systems [10,37]. 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 modeling 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* [36] and the *metabolic algorithm* developed and propagated by V. Manca and colleagues [6,29]. The first is directly based on mass action kinetics [18], whereas the latter considers a special form of competition of rules for objects, called the *mass partition principle*. Another approach has been proposed in [37], where rules have fixed reaction rates.

As has been discussed in [31], static dynamical P systems are Markov chains.

Problem 8. How can notions from the theory of dynamical systems [14], such as fixed points and attractors be defined for dynamical P systems?

Remark 8. This problem underlies much of the recent research on dynamical P systems. Indeed, one has to be careful here. The notion of a dynamical system is (notwithstanding proper generalizations [3]) that of a deterministic system, whereas dynamical P systems are stochastic systems.

One consequence of this difference is that the notion of a fixed point, so useful in the theory of deterministic systems, has little importance in the theory of dynamical P systems. By definition the only fixed points in a dynamical P system are sinks, i.e., halting states.

We now give a satisfying solution of this problem. The proper way to analyze dynamical P systems from the dynamical perspective is by considering a proper generalization of fixed points, *recurrent behavior*. Different notions of recurrence are discussed in [1], the most general being *chain-recurrence*, introduced by Conley. A point x in a dynamical system is chain-recurrent, if for all $\epsilon > 0$ and $T > 0$, there exists a finite sequence of states $x = x_0, x_1, x_2, \ldots, x_n = x$ from x to itself, and a corresponding finite sequence of times $t_0, \ldots, t_{n-1}$ in $[T, \infty)$, such that the distance between x_{i+1} and the endpoint t_i of the trajectory, starting at x_i and being followed for a time t_i, is less than ϵ for all i. In other words, a chain-recurrent point can be reached by a sequence that alternately (i) follows the dynamics for at least a time T, and (ii) jumps to a state within a distance ϵ. It can be shown that the chain recurrent set contains all fixed points, periodic points and limit sets.

In a dynamical P system, the state space has the discrete topology, and time evolution is also discrete. A chain-recurrent point then corresponds to a point that is reachable from itself, i.e., the chain recurrent set is exactly the set of *communicating* states. So in an irreducible P system, which consists of exactly one communicating class, the chain recurrent set is the whole state space. But note that each dynamical P system, when started from a single initial condition (or a different initial condition that comes from the same communicating class) that lies inside a communicating class, is irreducible. What makes the notion of invariant distribution interesting, is that it carries quantitative information about how often the system is expected to be in a certain state. States with a higher page rank will be visited more often than states with a lower page rank. The invariant distribution *orders* the recurrent states of the system by their importance.

From a stationary distribution we can derive a complexity measure for P systems that quantifies the complexity in dynamical behavior.

Definition 1. *The* entropy *of a P system is the entropy of its invariant probability distribution. That is, if $p : S \mapsto \mathbb{R}_+^n$ is its invariant distribution, then*

$$h = \frac{-\sum_{i \in S} p(i) \log p(i)}{\log |S|}$$

is its entropy.

The denominator has been chosen such that $0 \leq h \leq 1$ holds. A low value of h signifies simple dynamical behavior, whereas a value of h close to one is characteristic of random behavior.

This idea generalizes the *global entropy* of [11], 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 [9]). 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 relates to complexity considerations in other fields of science (confer [4,27]).

6 Approximating Asymptotic Behavior

Since the state space in dynamical P systems is usually infinite, a stationary distribution usually does not exist. 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, which is always finite.

Definition 2. *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 the the adjacency matrix of its object network, normalized row-wise such that its rows sum to one.

Definition 3. *The* ranking matrix *of a P system is the matrix* $\bar{\bar{C}}$ *(confer Eq. 1) where* C *is its connectivity matrix.*

Definition 4. *The* stationary object distribution *of a P system is the dominant eigenvector of its ranking matrix.*

The above definitions only use (i) the *topological* information about how objects can be transformed into each other. However, in a P system there are two more levels that can be considered, namely (ii) the stoichiometry, which introduces further constraints, and (iii) reaction rates. The latter has been discussed already, of course. However, incorporating the stoichiometry only, is not very satisfying. Eventually we need to come up with probabilities for a Markov chain, and although these can be readily defined from stoichiometric weights, this is a somewhat artificial construction that is difficult to interpret.

Let us finally, for completeness, consider the conventional analysis of steady state fluxes in biochemical networks [18]. 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 [23]. 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, corresponding to the complete probabilistic description of the system as in the beginning of the paper. The invariant distribution then induces a *unique* steady state flux (given by the product of the invariant distribution with the relative outdegrees), in contrast to the topological and the stoichiometric case. The implications of this, especially with regard to pathway analysis, have yet to be fully realized.

7 Page Ranking in P System Identification

In a previous work [30] 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 modeling and analysis process. Instead of analyzing a *given* P system, the problem is to *find* an interesting P system that then can be analyzed, 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 [22], aMAZE and KEGG [21]. For example, the LIGAND database [17], which is a particular database inside the KEGG repository, contains (as of version 42.0) information about 15053 chemical compounds (KEGG COMPOUND), 7522 biochemical reactions (KEGG REACTION) and 4975 enzymes (KEGG ENZYME) in ASCII text files that are easily parseable by computer.

In the usual approach [12,32] 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* [13].

A particular problem with this approach is the existence of so-called *currency* metabolites [19]. 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 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 [25], 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 specificity, i.e., a lower degree [12].

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.

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 [11].

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 recurrent dynamical behavior. In particular, we can now give operational definitions of the concept of "fixed points" for P systems as states with large invariant probability, whereas "transient" states will have very small invariant probability (on the order of α).

A different application has been in the identification of P system models from biochemical databases. 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 consists of more than 10000 vertices, the eigenvector calculation has to be done only once. The test of this idea is underway.

Acknowledgements. This work has been supported by the Dutch Science Foundation (NWO) under grant no. 635.100.006.

References

1. Alongi, J.M., Nelson, G.S.: Recurrence and Topology. American Mathematical Society (2007)
2. Altman, A., Tennenholtz, M.: Ranking systems: the PageRank axioms. In: Proc. 6th ACM conference on Electronic Commerce, pp. 1–8 (2005)
3. Arnold, L.: Random Dynamical Systems. Springer, Heidelberg (1998)
4. Badii, R., Politi, A.: Complexity. Hierarchical Structures and Scaling in Physics. Cambridge University Press, Cambridge (1997)
5. Bapat, R.B., Raghavan, T.E.S.: Nonnegative Matrices and Applications. Cambridge University Press, Cambridge (1997)
6. Bianco, L., Fontana, F., Manca, V.: P systems with reaction maps. Int. J. Found. Comp. Sci. 17, 3–26 (2006)
7. Brémaud, P.: Markov Chains. Gibbs Fields, Monte Carlo Simulation, and Queues. Springer, Heidelberg (1999)
8. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems 30, 107–117 (1998)
9. Chakrabarti, D., Faloutsos, C.: Graph mining: Laws, generators, and algorithms. ACM Computing Surveys 38, 1–69 (2006)
10. Ciobanu, G., Păun, G., Pérez-Jiménez, M.J. (eds.): Applications of Membrane Computing. Springer, Heidelberg (2006)
11. Cordón-Franco, A., Sancho-Caparrini, F.: A note on complexity measures for probabilistic P systems. J. Universal Computer Sci. 10, 559–568 (2004)
12. Croes, D., Couche, F., Wodak, S.J., van Helden, J.: Inferring meaningful pathways in weighted metabolic networks. J. Mol. Bio. 356, 222–236 (2006)
13. Epstein, E.: Finding the k shortest paths. SIAM J. Comput. 28, 652–673 (1998)
14. Guckenheimer, J., Holmes, P.: Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields. Springer, Heidelberg (1983)
15. Golub, G.H., Van Loan, C.F.: Matrix Computations. Johns Hopkins University Press (1996)
16. Google: Google Technology, `http://www.google.com/technology/`
17. Goto, S., Nishioka, T., Kanehisa, M.: LIGAND: chemical database for enzyme reactions. Bioinformatics 14, 591–599 (1998)
18. Heinrich, R., Schuster, S.: The Regulation of Cellular Systems. Springer, Heidelberg (1996)
19. Huss, M., Holme, P.: Currency and commodity metabolites: their identification and relation to the modularity of metabolic networks. IET Syst. Biol. 1, 280–285 (2007)
20. Kamvar, S., Haveliwala, T., Golub, G.: Adaptive methods for the computation of PageRank. Linear Algebra Appl. 386, 51–65 (2004)
21. Kanehisa, M., Araki, M., Goto, S., Hattori, M., et al.: KEGG for linking genomes to life and the environment. Nucleic Acids Research 484, D380–D484.(2008)
22. Karp, P.D., Riley, M., Saier, M., Paulsen, I.T., et al.: The EcoCyc and MetaCyc databases. Nucleic Acids Research 28, 56–59 (2000)
23. Kauffman, K.J., Prakash, P., Edwards, J.S.: Advances in flux balance analysis. Current Opinion in Biotechnology 14, 491–496 (2003)
24. Klipp, E., Herwig, R., Kowald, A., Wierling, C., et al.: Systems Biology in Practice. Wiley-VCH, Chichester (2005)
25. Kuffner, R., Zimmer, R., Lengauer, T.: Pathway analysis in metabolic databases via differential metabolic display (DMD). Bioinformatics 16, 825–836 (2000)

26. Langville, A.N., Meyer, C.D.: Deeper inside PageRank. Internet Math. 1, 335–380 (2004)
27. Lind, D., Marcus, B.: An Introduction to Symbolic Dynamics and Coding. Cambridge University Press, Cambridge (1995)
28. Ma, N., Guan, J., Zhao, Y.: Bringing PageRank to the citation analysis. Information Processing & Management 44, 800–810 (2008)
29. Manca, V., Bianco, L.: Biological networks in metabolic P systems. BioSystems 91, 489–498 (2008)
30. 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 Proc., pp. 46–49 (2008)
31. Muskulus, M., Besozzi, D., Brijder, R., Cazzaniga, P., et al.: Cycles and communicating classes in membrane systems and molecular dynamics. Theoretial Computer Sci. 372, 242–266 (2007)
32. 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 (in press)
33. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking. Bringing order to the web. Technical Report, Stanford University (1998), `http://dbpubs.stanford.edu:8090/pub/1999-66`
34. Păun, G.: Computing with membranes. J. Computer System Sci. 61, 108–143 (2000)
35. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
36. Pescini, D., Besozzi, D., Mauri, G., Zandron, C.: Dynamical probabilistic P systems. Intern. J. Found. Comp. Sci. 17, 183–204 (2006)
37. Romero-Campero, F.J., Pérez-Jiménez, M.J.: Modelling gene expression control using P systems. The Lac operon, a case study. Biosystems 91, 438–457 (2008)
38. Vise, D., Malseed, M.: The Google Story. Random House (2006)

First Steps Towards a Wet Implementation for τ-DPP

Dario Pescini, Paolo Cazzaniga,
Claudio Ferretti, and Giancarlo Mauri

Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336, 20126 Milano, Italy
{pescini,cazzaniga,ferretti,mauri}@disco.unimib.it

Abstract. 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 register machines instructions. Finally will discuss how this computing components can be composed in a more complex chemical computing system, with a structure based on the membrane structure of τ-DPP, to move toward a wet implementation using the micro reactors technology.

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 others inspired by the structure and functioning of living cells, have been proposed. One of these models, introduced in [14], is called P systems (or membrane systems). The basic definition of P systems consists of a hierarchical structure composed by several compartments, each one delimited by a membrane. Inside every compartment, a set of evolution rules and a multiset of objects are placed. The rules – precisely, multiset rewriting rules – are used to describe the modification and the communication among the membranes of the objects occurring inside the system. In particular, the current system state is represented by means of objects quantities.

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 introduced by Gillespie in [7]. In particular, τ-DPP extends the tau-leaping procedure [4] in order to quantitatively simulate the behavior of complex biological and chemical systems, embedded in membrane structures composed by different volumes.

The aim of this work is to implement the computational model based on chemical reacting systems using micro reactors. Micro reactors [9] are laboratory

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 355–373, 2009.

devices consisting of several reacting volumes (reactors) with a size at the scale of the μl, connected by channels used to transport molecules.

Hence, in this paper we will establish a correspondence between τ-DPP and chemical reacting systems occurring inside micro reactors. There is a close relation between the topological description of the two systems: they are both composed by several volumes, and among these volumes it is possible to communicate molecules. Moreover, the approach based on multiset rewriting rules, that characterizes τ-DPP, is similar to the chemical reacting process occurring within a micro reactor. Furthermore, both τ-DPP and chemical reacting systems emphasize the intrinsic stochasticity of chemical processes. The "noise", associated to the stochastic behavior, rules the system dynamics at the micro-scales. At this scale, the small volumes and the high dilutions result in a system where 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 build a feasible wet implementation of P systems (in particular, of τ-DPP). To obtain a description of τ-DPP that can be implemented using micro reactors in a straightforward way, we consider the encoding of Boolean functions and register machine instructions through chemical reactions, following the *chemical computing principles*.

Chemical computing [6] is a technique used to process information by means of real molecules, modified by chemical reactions, or by using electronic devices that are programmed following some principles coming from chemistry. Moreover, in chemical computing, the result of a computation is represented by the emergent global behavior, obtained from the application of small systems characterized by chemical reactions.

Exploiting the chemical organization theory [12], we can define a chemical network in order to describe a system as a collection of reactions applied to a given set of molecular species. Moreover, we can identify the set of (so called) organizations, in the set of molecular species, to describe the behavior of such system. So doing, the behavior is traced by means of "movement" between organizations.

Furthermore, a different kind of problem encoding will be 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 realize the instructions of the register machines. For instance, in Section 4, the formalization and the simulation of a decrement instruction by means of τ-DPP, is presented.

The paper is organized as follows: in Section 2, membrane systems and its variant of τ-DPP are explained. The chemical computing framework and chemical organization theory are presented in Section 3. In Section 4, we show the results of the simulations of small components, such as the NAND logic circuit 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 [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 that can be handled by means of τ-DPP, is not only related to the number of reactions (rules) and species (objects) involved, but it results also from the topological structure of the system, that can be composed by many volumes.

2.1 Membrane Systems

P systems, or membrane systems, have been 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 topological and hierarchical organization of a system consisting of distinct compartments. The definition of membrane structure is given through a set of membranes with a distinct label (usually 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.

In particular, each membrane identifies a *region*, delimited by the membrane itself and any other adjacent membrane possibly present inside it. The number of membranes in a membrane structure is called the *degree* of the P system. 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 alphabet V. In order to denote the presence of multiple copies of objects inside the membranes, multisets are usually used.

The objects inside the membranes of a P system are transformed by means of *evolution rules*. These are multiset rewriting rules of the form $r_i : u \to v$, where u and v are multisets of objects. The meaning of the generic rule i is that the multiset u is modified into the multiset v. There exists a special symbol δ used to dissolve (namely, remove) the membrane where the rule is applied together with its set of rules. In this paper we will not make use of the dissolving operation.

Moreover, it is possible to associate a target to v, representing the membrane where the multiset v 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 (usually, this target label is omitted in the systems description). 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. Finally, if the target is in_j, where j is a label of a

membrane, then the object is sent into the membrane labeled with j. It is possible to apply this kind of rule, only if the membrane j is placed immediately inside the membrane where the rule is executed.

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 can be 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 previously specified *output membrane* or sent 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 system satisfies this requirement, hence, the spatial arrangement of P system is exploited in the τ-DPP description. In particular, τ-DPP has been defined starting from a variant of P systems called dynamical probabilistic P systems (DPP). DPP were introduced in [18]: they exploit the membrane structure of P systems and they associate probabilities with the rules, 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 these two membrane systems variants: DPP provides only a qualitative description of the analyzed system, that is, "time" is not associated to the evolution steps, while τ-DPP is able to give a quantitative description tracing the time-stream of the evolution.

The τ-DPP approach is designed to share a common time increment among all the membranes, used to accurately extract the rules that will be executes in each compartment (at each step). 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 behavior 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 behavior 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 behavior of the system, the tau-leaping method describes an approximated behavior with respect to the CME, but it is faster for what concerns the computational time required.

To describe the correct behavior of the whole system, 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 time step length is the following. Each membrane independently computes a candidate time increment (exploiting the tau-leaping procedure), based on its internal state. The smallest time increment among all membranes 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 *synchronized* at the end of each iterative step, when all the rules are executed.

The modified tau-leaping procedure of τ-DPP is also used to select the set of rules that will be executed during the current leap. This is done locally, that is, each membrane selects the kind of evolution it will follow, independently from other volumes. 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

$$\Upsilon = (V_0, \ldots, V_n, \mu, \mathcal{S}, M_0, \ldots, M_n, R_0, \ldots, R_n, C_0 \ldots C_n),$$

where:

- $V_0, \ldots, V_n$ 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;
- $M_0, \ldots, M_n$, are the sets of multisets over $\mathcal{S}$ occurring inside the membranes $V_0, \ldots, V_n$, representing the internal state of the volumes. The multiset M_i $(0 \leq i \leq n)$ is defined over S^*;

- $R_0, \ldots, R_n$ are the sets of rules defined in volumes $V_0, \ldots, V_n$, respectively. A rule can be of internal or of communication type (as described below);
- $C_0, \ldots, C_n$ are the sets of stochastic constants associated to the rules defined in volumes $V_0, \ldots, V_n$.

Inside the volumes of a system described by means of τ-DPP, two kinds of evolution rules can be placed. These are called *internal* and *communication* rules. Internal rules describe the evolution of objects that remain in the region where the rule is executed (i.e. target *here*). Communication rules send objects from the membrane where they are applied to an adjacent volume(i.e. target in_j or *out*). Moreover, they can also modify the objects during the communication process.

The sets of stochastic constants $C_0, \ldots, C_n$, associated to the sets of rules $R_0, \ldots, R_n$, 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 $\alpha_1 X_1 + \alpha_2 X_2 + \ldots + \alpha_k X_k \rightarrow (\beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k, target)$, where $X_1, \ldots, X_k \in \mathcal{S}$ are the molecular species and $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k \in \mathbb{N}$ represent the multiplicities of the objects involved in the rule. 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 a reaction involving more than three objects is close to zero.

The target of the rules follows the same definition given for the membrane systems in Section 2.

There is a difference in the application of internal and communication rules during the computation of the time increment (τ). In the procedure use to compute τ, while for internal rules both left-hand and right-hand sides are involved, for communication rules only the left-hand side is involved. This distinction is needed because the right-hand side of internal and communication rules is differently used to update the system state. For internal rules the right-hand side modifies the membrane where the rule is applied, whereas for communication rules it 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 iterative step, and will be therefore considered to determine the state of the target volume for the next iteration.

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 ($i = 0, \ldots, n$) 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. Initialization: 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, $\mu = 1, \ldots, l$, and evaluate the sum of all the propensity functions in V_i, $a_0 = \sum_{\mu=1}^{l} a_\mu$. 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 $\tau_{min} = \min\{\tau_0, \ldots, \tau_n\}$ among those generated independently inside all volumes $V_0, \ldots, V_n$, 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 $\tau_{min} = \min\{\tau_0, \ldots, \tau_n\}$ 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 $\tau_i = \tau_{SSA}$ 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 $\tau_i = \tau_{nc1c}$ 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 $\tau_i = \tau_{nc1c}$ 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 to` *step 2*, otherwise `go to` *step 9*.

Step 9. Set $\tau_i = \tau_{SSA}$ for the next iteration, then `go 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 to` *step 2*.

The algorithm begins loading the initial conditions of the membrane. The next operation is the computation of the propensity functions (and their sum a_0) 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 synchronize with them; then, it 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.

On the other hand, 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 itself, 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 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 will be the result of the application of *step 7*.

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. Otherwise, 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 (from the set of non-criticals) to apply in the current iteration.

The last step is the system update. Here every membrane executes the selected rules and updates 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 synchronize the volumes.

3 Chemical Computing

In this section we introduce the basic notions of chemical computing, a novel computational paradigm where the information is processed by means of chemical reactions. In particular, starting from the chemical computing field, we recall the basic definitions of chemical organization theory, used to analyze chemical computing systems in order to obtain useful knowledge, such as the emergent behavior of the studied system.

Biological systems are characterized by different mechanisms, employed in their evolution, that make them able to process information. These characteristics are: robustness, self-organization, concurrency, fault-tolerance and evolvability. The global information process comes by using, inside the biological system, 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 both real molecules and electronic devices, programmed using principles taken from chemistry.

In general, the analysis of the solutions of chemical reaction processes is hard because of their nonlinearity. The same problems are related to the analysis of biological systems since the behavior of local parts can be very different from the global behavior.

In order to work out this problem, the notions of chemical organization theory can be used to obtain the emergent behavior of the system, starting from its small components, hence linking the evolution governed by every single reaction with the global dynamics of the system.

Chemical organization theory [12] is used to identify a hierarchy of self maintaining sub-networks, belonging to a chemical reaction network. These subnetworks are called organizations. In particular, a chemical organization is a set of molecular species that satisfies two properties, that is, it is algebraically closed and stoichiometrically self-maintaining. Here we report an informal definition of these concepts, and refer the reader to [12] for formal definitions and further details.

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 $\alpha_1 m_1 + \alpha_2 m_2 + \ldots + \alpha_k m_k \rightarrow \beta_1 m_1 + \beta_2 m_2 + \ldots + \beta_k m_k$, 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.

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 organizations of a chemical network, only stoichiometric information (set of rules) is needed.

The set of organizations of a chemical network can be exploited to describe the dynamics of the system, by means of the movement among different organizations. Namely, the dynamics is traced looking at the system state and at the organizations "represented" by the molecules occurring in the system. Therefore, 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). Note that, only the algebraic analysis of chemical organizations is sufficient in order to obtain this behavior. Furthermore, the behavior of the

system 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 reaction networks to compute, we need to assume that a computational problem can be described as a Boolean function, which in turn can be computed as a composition of many simple functions (e.g. the binary NAND). Therefore, we will create a reaction network (called Boolean network), based on a set of Boolean functions and Boolean variables.

Consider a set of M Boolean functions $F_1, \ldots, F_M$ 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 M 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.

Given a Boolean network (as described above), the associated reaction network $\langle \mathcal{M}, \mathcal{R} \rangle$, as presented in [12], is defined as follows. For each Boolean variable b_j, two different molecular species, representing the values 0 and 1 of the variable, are added to $\mathcal{M}$. 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).

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 initialized because they are not set by the Boolean functions. The initialization is encoded by means of inflow reactions. These reactions are zero-order reactions producing molecules from the empty set.

4 Definition and Simulation of *Component Reaction Networks* Using τ-DPP

To lay out our path from a model of computation to a chemical computing device, we define and simulate test case τ-DPP systems using techniques inspired by the literature on reaction systems [6,12]. Those simple systems must be powerful enough to compute, when assembled in more complex combination, 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 chemical reactions. We then present some simulation results of our systems.

We recall that *register machines* [13] are universal abstract computing devices, where a finite set of uniquely labeled instructions is given, and which keep updated a finite set of registers at any time (holding integer numbers) by performing a sequence of instructions, chosen according to their labels. Every instruction can be of one of the following, here informally introduced:

- `ADD`: a specified register is increased by 1, and the label of next instruction is nondeterministically chosen between two labels specified in the last instruction applied,
- `SUB`: a specified register is checked, and if it is non-empty, then 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.

4.1 The NAND and XOR Logic Circuits

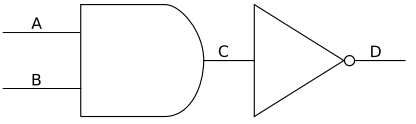
The NAND logic circuit has been implemented with the sequential composition of an AND and a NOT gate as shown in Figure 1 (left). Following the chemical computing guidelines described in Section 3, we define the logic circuit with the rules listed in Figure 1 (right). Rules $r_1, \ldots, r_4$ compute the AND function, rules $r_5, \ldots, r_6$ compute the NOT function and rules $r_7, \ldots, r_{10}$ "clean" the system when both values of a variable are present at the same time, as described in Section 3. Finally, rules $r_{11}, \ldots, r_{14}$ represent the inputs of the gate because they produce the molecules a, A, b and B, representing the inputs $A = 0$, $A = 1$, $B = 0$ and $B = 1$ of the NAND logic circuit, respectively. 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 because molecules a and b are produced. The rationale behind this, is that the different inputs for the system are obtained producing the molecular species used to represent that particular values. The values of the costants reported in the table have been used to perform the simulation of the NAND behavior by means of τ-DPP.

Starting from the set of rules presented above for the NAND logic circuit, it is 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 logic circuit;
- μ is the membrane structure $[_0\ ]_0$;
- $\mathcal{S} = \{a, A, b, B, c, C, d, D\}$ is the set of molecular species;
- $M_0 = \{a^{m_a}, A^{m_A}, b^{m_b}, B^{m_B}, c^{m_c}, C^{m_C}, d^{m_d}, D^{m_D}\}$, is the set of multisets occurring inside the volume V_0. m_a, m_A, m_b, m_B, m_c, m_C, m_d and $m_D \in \mathbb{N}$;



Reaction	Constant
$r_1: \ a + b \to c$	$c_1 = 1 \cdot 10^{-3}$
$r_2: \ a + B \to c$	$c_2 = 1 \cdot 10^{-3}$
$r_3: \ A + b \to c$	$c_3 = 1 \cdot 10^{-3}$
$r_4: \ A + B \to C$	$c_4 = 1 \cdot 10^{-3}$
$r_5: \ c \to D$	$c_5 = 1 \cdot 10^{-2}$
$r_6: \ C \to d$	$c_6 = 1 \cdot 10^{-2}$
$r_7: \ a + A \to \lambda$	$c_7 = 1 \cdot 10^{-1}$
$r_8: \ b + B \to \lambda$	$c_8 = 1 \cdot 10^{-1}$
$r_9: \ c + C \to \lambda$	$c_9 = 1 \cdot 10^{-1}$
$r_{10}: d + D \to \lambda$	$c_{10} = 1 \cdot 10^{-1}$
$r_{11}: \lambda \to a$	$c_{11} \in \{1, 0\}$
$r_{12}: \lambda \to A$	$c_{12} \in \{1, 0\}$
$r_{13}: \lambda \to b$	$c_{13} \in \{1, 0\}$
$r_{14}: \lambda \to B$	$c_{14} \in \{1, 0\}$

Fig. 1. The NAND logic circuit (left) and the set of reactions used to implement it (right)

- $R_0 = \{r_1, \ldots, r_{14}\}$ is the set of rules defined in volume V_0 and reported in Table 1. Due to the membrane structure μ, all the rules here involved are internal.
- $C_0 = \{c_1, \ldots, c_{14}\}$ is the set of stochastic constants associated to the rules defined in R_0, and reported in Table 1.

In Figure 2, the result of the simulation of the NAND gate is 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 is formulated as 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 are change from a, B to A, B, setting c_{11} and c_{14} to 0 and c_{12} and c_{13} to 1. 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.

The XOR logic circuit (see Figure 3 (left)) has been implemented using the set of rules listed in Figure 3 (right). The rules $r_1, \ldots, r_4$ compute the XOR function and $r_5, \ldots, r_7$ "clean" the system when both values of a variable are present at the same time, as described in Section 3. Finally, 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 XOR gate is 0 for both the input lines. The values of the constant reported in the table have been used to perform the simulation by means of τ-DPP.

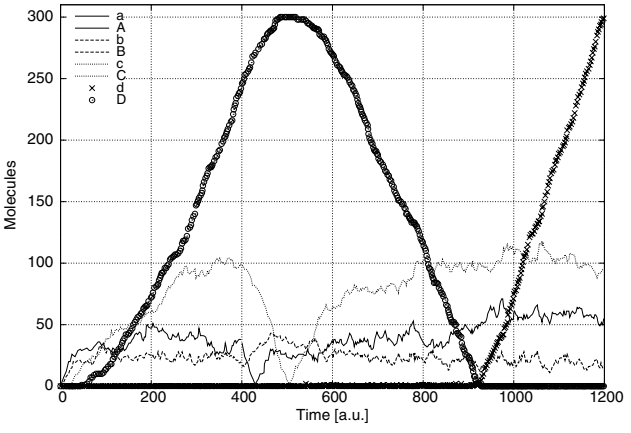
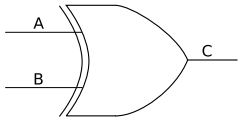


Fig. 2. Plot of the dynamics of the NAND unit with two inputs in succession. The initial multiset is 0 for all the molecular species.



Reaction	Constant
$r_1: \ a + b \to c$	$c_1 = 1 \cdot 10^{-3}$
$r_2: \ a + B \to C$	$c_2 = 1 \cdot 10^{-3}$
$r_3: \ A + b \to C$	$c_3 = 1 \cdot 10^{-3}$
$r_4: \ A + B \to c$	$c_4 = 1 \cdot 10^{-3}$
$r_5: \ a + A \to \lambda$	$c_5 = 1 \cdot 10^{-1}$
$r_6: \ b + B \to \lambda$	$c_6 = 1 \cdot 10^{-1}$
$r_7: \ c + C \to \lambda$	$c_7 = 1 \cdot 10^{-1}$
$r_8: \ \lambda \to a$	$c_8 \in \{1, 0\}$
$r_9: \ \lambda \to A$	$c_9 \in \{1, 0\}$
$r_{10}: \lambda \to b$	$c_{10} \in \{1, 0\}$
$r_{11}: \lambda \to B$	$c_{11} \in \{1, 0\}$

Fig. 3. The XOR logic circuit (left), and set of reactions used to implement it (right)

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 logic circuit;
- μ is the membrane structure $\left[_0 \ \right]_0$;
- $\mathcal{S} = \{a, A, b, B, c, C, d, D\}$ 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. m_a, m_A, m_b, m_B, m_c, and $m_C \in \mathbb{N}$;
- $R_0 = \{r_1, \ldots, r_{11}\}$ is the set of rules defined in volumes V_0 and reported in Table 3. Due to the membrane structure μ, all the rules here involved are internal.
- $C_0 = \{c_1, \ldots, c_{11}\}$ is the set of stochastic constants associated to the rules defined in V_0 and reported in Table 3.

In Figure 4, the result of the simulation of the XOR gate is 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 is formulated as 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, setting c_8 to 0 and c_9 to 1. 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.

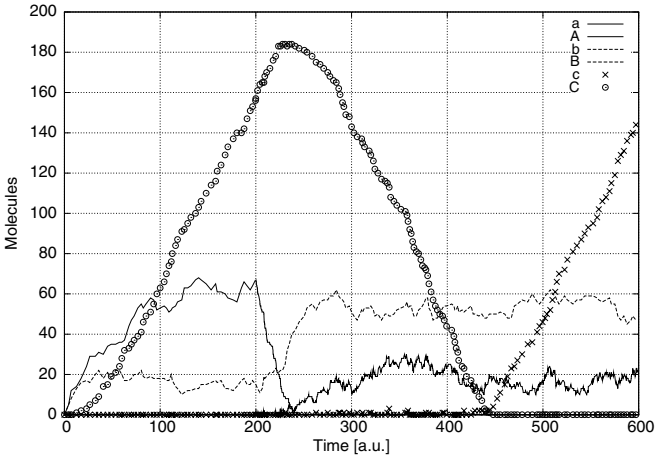


Fig. 4. Plot of the dynamics of the XOR unit with two inputs in succession. The initial multiset is 0 for all the molecular species.

4.2 The SUB Instruction

We now describe and simulate a τ-DPP composed by 2 volumes reproducing, by means of chemical reactions operating on a set of molecular species, the behavior of a `SUB` instruction of a register machine. This type of instruction is important because it hides a conditional behavior, checking whether a register is zero or not, respectively choosing a different label for the next instruction. The availability of conditional instructions is a key issue in computing devices.

We implement a system where the quantity stored inside the register is represented by the amount of objects u occurring in volume V_1. The label for the next instruction in related to the production of objects p or z in volume V_0. Finally, to start the system, objects s are produced inside V_0. s' and z' are additional molecular species used to implement the instruction.

In order to correctly execute the SUB instruction, when molecules s are sent inside the volume V_1, the system first checks if the register value is zero, that is, if any u molecule occurs inside V_1, then, the label for the next instruction is produced.

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 $[_0\ [_1\ \ ]_1\ ]_0$;
- $\mathcal{S}_0 = \{p, s, s', z\}$ and $\mathcal{S}_1 = \{p, s, u, z, z'\}$ are the sets of molecular species of volumes V_0 and V_1, respectively;
- $M_0 = \{p^{m_p}, s^{m_s}, s'^{m_{s'}}, z^{m_z}\}$, $M_1 = \{p^{m_p}, s^{m_s}, u^{m_u}, z^{m_z}, z'^{m_{z'}}\}$, are the multisets occurring inside the membranes V_0 and V_1, respectively. m_p, m_s, $m_{s'}$, m_z, m_u and $m_{z'} \in \mathbb{N}$;
- $R_0 = \{r_{0_1}, \ldots, r_{0_5}\}$, $R_1 = \{r_{1_1}, \ldots, r_{1_3}\}$ are the sets of rules defined in volumes V_0, V_1, respectively, and reported in Table 1;
- $C_0 = \{c_{0_1}, \ldots, c_{0_5}\}$, $C_1 = \{c_{1_1}, \ldots, c_{1_3}\}$ is the sets of stochastic constants associated to the rules defined in V_0 and V_1, respectively, and reported in Table 1.

Table 1. Reactions 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	**Constant**
$r_{0_1} : 2p \to (p, here)$	$c_{0_1} = 1$
$r_{0_2} : z + p \to (z, here)$	$c_{0_2} = 1$
$r_{0_3} : 2z \to (z, here)$	$c_{0_3} = 1$
$r_{0_4} : s \to (s, in_1)$	$c_{0_4} = 1$
$r_{0_5} : s' \to (s, here)$	$c_{0_5} = 6 \cdot\ 10^{-2}$

Reaction	**Constant**
$r_{1_1} : s + u \to (p, out)$	$c_{1_1} = 1 \cdot 10^3$
$r_{1_2} : s + z \to (z + z', here)$	$c_{1_2} = 1$
$r_{1_3} : z' \to (z, out)$	$c_{1_3} = 1$

The simulation starts with a positive register value within V_1, represented by the u molecules; the system receives a sequence of SUB requests, due to the presence of s' molecules in V_0, transformed in s by the application of rule r_{0_5} and then sent to V_1 by rule r_{0_4}. Figure 5 shows the two execution phases: in the first phase the counter is decremented, as long as there are s' molecules available in V_0, and objects p are produced in V_0. Afterwards, when the register counter reaches zero (all u molecules are consumed), only objects z are produced in V_0.

This system is initialized 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 behavior occurring when the simulated register goes to zero.

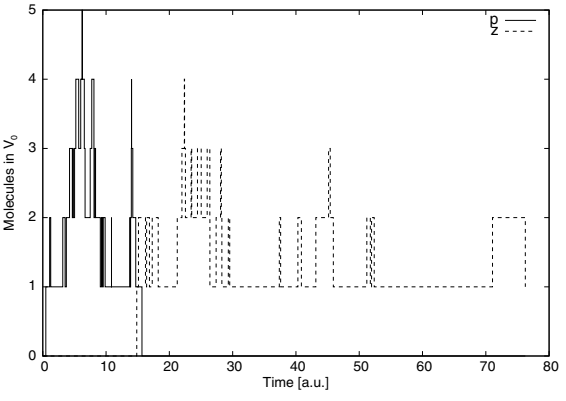


Fig. 5. Plot of the dynamics of the SUB unit

4.3 The SUBADD Module

The SUB unit can be extended to perform both a `SUB` and an `ADD` instructions, according to the object received from the environment: s or a, respectively. The register value is stored inside volume V_2 and it is represented by the amount of molecules u occurring inside of it. The rules shown in Table 2 are defined to perform both operations, and the choice of molecular species avoid mixing them. In particular, rules $r_{0_1}, \ldots, r_{0_4}$, $r_{1_1}, \ldots, r_{1_6}$ and $r_{2_1}, \ldots, r_{2_3}$ define the SUB instruction (checking whether the register value is zero or not). The other rules are used to perform the ADD instruction.

The τ-DPP Υ_{SUBADD} implementing the SUBADD module 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;
- μ is the nested membrane structure $[_0\ [_1\ [_2\ ]_2\ ]_1\ ]_0$;
- $\mathcal{S}_0 = \{l, l', s, z, m, n, p, k, k', a, A, o, q\}$, $\mathcal{S}_1 = \{s, p, z, a, A\}$ and $\mathcal{S}_2 = \{s, u, z, z', a, a'\}$ are the sets of molecular species;
- $M_0 = \{l^{m_l}, l'^{m_{l'}}, s^{m_s}, z^{m_z}, m^{m_m}, n^{m_n}, p^{m_p}, k^{m_k}, k'^{m_{k'}}, a^{m_a}, A^{m_A}, o^{m_o}, q^{m_q}\}$, $M_1 = \{s^{m_s},\ p^{m_p}, z^{m_z}, a^{m_a}, A^{m_A}\}$ 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. m_l, $m_{l'}$, m_m, m_k, $m_{k'}$, m_o, m_q, m_s, m_p, m_z, m_a, m_A, m_u, $m_{z'}$ and $m_{a'} \in \mathbb{N}$;
- $R_0 = \{r_{0_1}, \ldots, r_{0_8}\},\ R_1 = \{r_{1_1}, \ldots, r_{1_8}\},\ R_2 = \{r_{2_1}, \ldots, r_{2_5}\}$ are the sets of rules defined in volumes V_0, V_1 and V_2 respectively, and reported in Table 2;
- $C_0 = \{c_{0_1}, \ldots, c_{0_8}\},\ C_1 = \{c_{1_1}, \ldots, c_{1_8}\},\ C_2 = \{c_{2_1}, \ldots, c_{2_5}\}$ are the sets of stochastic constants associated to the rules defined in V_0, V_1 and V_2, respectively, and reported in Table 2.

The results of the simulations are shown in Figure 6.

Table 2. Reactions 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	Constant
$r_{0_1} : l \to (l' + s, here)$	$c_{0_1} = 1$
$r_{0_2} : s \to (s, in_1)$	$c_{0_2} = 1$
$r_{0_3} : l' + z \to (n, here)$	$c_{0_3} = 1$
$r_{0_4} : l' + p \to (m, here)$	$c_{0_4} = 1$
$r_{0_5} : k \to (k' + a, in_1)$	$c_{0_5} = 1$
$r_{0_6} : a \to (a, in_1)$	$c_{0_6} = 1$
$r_{0_7} : k' + A \to (o, here)$	$c_{0_7} = 1$
$r_{0_8} : k' + A \to (q, here)$	$c_{0_8} = 1$

Reaction	Constant
$r_{1_1} : s \to (s, in_2)$	$c_{1_1} = 1$
$r_{1_2} : 2p \to (p, here)$	$c_{1_2} = 1$
$r_{1_3} : 2z \to (z, here)$	$c_{1_3} = 1$
$r_{1_4} : p + z \to (p, here)$	$c_{1_4} = 1$
$r_{1_5} : z \to (z, out)$	$c_{1_5} = 1$
$r_{1_6} : p \to (p, out)$	$c_{1_6} = 1$
$r_{1_7} : a \to (a, in_2)$	$c_{1_7} = 1$
$r_{1_8} : A \to (A, out)$	$c_{1_8} = 1$

Reaction	Constant
$r_{2_1} : s + u \to (p, out)$	$c_{2_1} = 1 \cdot 10^3$
$r_{2_2} : s + z \to (z + z', here)$	$c_{2_2} = 1$
$r_{2_3} : z' \to (z, out)$	$c_{2_3} = 1$
$r_{2_4} : a \to (u + a', here)$	$c_{2_4} = 1$
$r_{2_5} : a' \to (A, out)$	$c_{2_5} = 1$

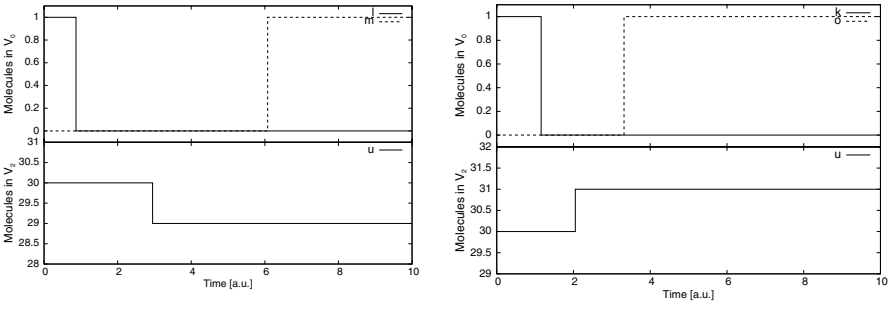


Fig. 6. Plot of the dynamics of the SUBADD module performing a decrement instruction (left) and an increment instruction (right) on a register

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, as well as the general reaction network considered in literature, often 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 be reproduced with a tree-like structure of nested membranes, communicating between adjacent ones. To avoid this limitation, we could move our studies to other variants of P systems, which allow more free adjacency relations between membranes, such as "Tissue P

Systems" [11]. In particular, this P systems variant can be exploited to represent general micro reactors grids.

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, enclosing 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 inside the skin membrane the molecules representing the current instruction label. For instance, if instruction l increments register r, then the rules would be defined to produce objects a_r and send them to an internal membrane representing register r. That internal membrane will then 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. Additional details related to the halting of the computation need to be specified.

This approach to the implementation of complex systems leads to some open problems worth being studied. How does the passage from single simple components to complete universal devices, with the required connectivity, scale? It is well known that small universal register machines can be built, as shown in [10], but their τ-DPP implementation, and eventually their chemical system implementation have to be evaluated.

Moreover, the computational efficiency of these systems can be studied, 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. Note that, the stochasticity of τ-DPP has to be considered in the computational complexity study.

References

1. Besozzi, D., Cazzaniga, P., Pescini, D., Mauri, G.: Modelling metapopulations with stochastic membrane systems. Biosystems 91, 499–514 (2008)
2. Besozzi, D., Cazzaniga, P., Pescini, D., Mauri, G.: Seasonal variance in P system models for metapopulations. Progress in Natural Science 17, 392–400 (2007)
3. Berry, G., Boudol, G.: The chemical abstract machine. Theoretical Computer Sci. 96, 217–248 (1992)
4. Cao, Y., Gillespie, D.T., Petzold, L.R.: Efficient step size selection for the tau-leaping simulation method. Journal Chemical Physics 124, 44109 (2006)
5. Cazzaniga, P., Pescini, D., Besozzi, D., Mauri, G.: Tau leaping stochastic simulation method in P systems. In: Hoogeboom, H.J., et al. (eds.) WMC 2006. LNCS, vol. 4361, pp. 298–313. Springer, Heidelberg (2006)
6. Dittrich, P.: Chemical computing. In: Banâtre, J.-P., Fradet, P., Giavitto, J.-L., Michel, O. (eds.) UPP 2004. LNCS, vol. 3566, pp. 19–32. Springer, Heidelberg (2005)
7. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journal Physical Chemistry 81, 2340–2361 (1977)

8. Gillespie, D.T., Petzold, L.R.: Approximate accelerated stochastic simulation of chemically reacting systems. Journal Chemical Physics 115, 1716–1733 (2001)
9. Haswell, S.J., Middleton, R.J., O'Sullivan, B., Skelton, V., Watts, P., Styring, P.: The application of microreactors to synthetic chemistry. Chemical Communication, 391–398 (2001)
10. Korec, I.: Small universal register machines. Theoretical Computer Sci. 168, 267–301 (1996)
11. Martín-Vide, C., Păun, G., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Sci. 296, 295–326 (2003)
12. Matsumaru, N., Centler, F., Speroni di Fenizio, P., Dittrich, P.: Chemical organization theory as a theoretical base for chemical computing. Intern. J. Unconventional Computing 3, 285–309 (2005)
13. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
14. Păun, G.: Computing with membranes. J. Computer and System Sci. 61, 108–143 (2000)
15. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
16. Pescini, D., Besozzi, D., Mauri, G., Zandron, C.: Dynamical probabilistic P systems. Intern. J. Foundations of Computer Sci. 17, 183–204 (2006)
17. Pescini, D., Besozzi, D., Mauri, G.: Investigating local evolutions in dynamical probabilistic P systems. In: Proc. Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2005), pp. 440–447. IEEE Computer Society Press, Los Alamitos (2005)
18. Pescini, D., Besozzi, D., Zandron, C., Mauri, G.: Analysis and simulation of dynamics in probabilistic P systems. In: Carbone, A., Pierce, N.A. (eds.) DNA 2005. LNCS, vol. 3892, pp. 236–247. Springer, Heidelberg (2006)

Defining and Executing P Systems with Structured Data in K*

Traian Şerbănuţă, Gheorghe Ştefănescu, and Grigore Roşu

Department of Computer Science, University of Illinois at Urbana-Champaign
201 N. Goodwin, Urbana, IL 61801
{tserban2,stefanes,grosu}@cs.uiuc.edu

Abstract. 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, e.g., 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, computation calculi, etc. 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

* Supported in part by NSF grants CCF-0448501, CNS-0509321 and CNS-0720512, by NASA contract NNL08AA23C, by the Microsoft/Intel funded Universal Parallel Computing Research Center at UIUC, and by several Microsoft gifts.

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 374–393, 2009.

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. 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,[1] 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, too. 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. The objects may be used in a unique huge soup and the evolution rules, previously used in

[1] Lists can be encoded with (multi)sets, but allowing them as "first-class citizens" gives the K user the capability to efficiently match fragments of lists.

specific regions, become global rules, but applied to similar path-tagged objects. This representation 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 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 of [23] for the current version of the draft book) using as an example a simple concurrent language. We start with the simple imperative language IMP defined in Tables 1 and 2. Then, we extend IMP with threads and call the resulting language CIMP.

Table 1. K-annotated syntax of IMP

Int ::=	... all integer numbers		
$Bool$ ::=	$\mathsf{true} \mid \mathsf{false}$		
$Name$ ::=	all identifiers; to be used as names of variables		
Val ::=	Int		
$AExp$ ::=	$Val \mid Name$		
	$AExp + AExp$	$[strict,\ extends +_{Int\times Int\to Int}]$	(r_1)
$BExp$ ::=	$Bool$		
	$AExp \leq AExp$	$[seqstrict,\ extends\ \leq_{Int\times Int\to Bool}]$	(r_2)
	$\mathsf{not}\ BExp$	$[strict,\ extends\ \neg_{Bool\to Bool}]$	(r_3)
	$BExp\ \mathsf{and}\ BExp$	$[strict(1)]$	(r_4)
$Stmt$ ::=	$Stmt; Stmt$	$[s_1; s_2 = s_1 \curvearrowright s_2]$	(r_5)
	$Name := AExp$	$[strict(2)]$	(r_6)
	$\mathsf{if}\ BExp\ \mathsf{then}\ Stmt\ \mathsf{else}\ Stmt$	$[strict(1)]$	(r_7)
	$\mathsf{while}\ BExp\ \mathsf{do}\ Stmt$		(r_8)
	$\mathsf{halt}\ AExp$	$[strict]$	(r_9)
Pgm ::=	$Stmt; AExp$		

Table 2. K configuration and semantics of IMP

$KResult ::= Val$	$\langle\!\langle \underset{\cdot}{x := v} \rangle\!\rangle_k\ \langle\!\langle (x, \underset{v}{\_}) \rangle\!\rangle_{state}$
$K ::= KResult \mid \mathsf{List}^{\curvearrowright}[K]$	$\mathsf{if}\ \mathsf{true}\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2 \to s_1$
$Config ::= \langle\!\langle K \rangle\!\rangle_k \mid \langle\!\langle \mathsf{Set}[(Name, Val)] \rangle\!\rangle_{state}$	$\mathsf{if}\ \mathsf{false}\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2 \to s_2$
$\mid\ \langle\!\langle \mathsf{Set}[Config] \rangle\!\rangle_{\top}$	$\langle\!\langle \underline{\quad \mathsf{while}\ b\ \mathsf{do}\ s \quad} \rangle\!\rangle_k$
	$\mathsf{if}\ b\ \mathsf{then}\ (s; \mathsf{while}\ b\ \mathsf{do}\ s)\ \mathsf{else}\ \cdot$
$\langle\!\langle \underset{v}{x} \rangle\!\rangle_k\ \langle\!\langle (x, v) \rangle\!\rangle_{state}$	$\langle\!\langle \mathsf{halt}\ i \rangle\!\rangle_k \to \langle\!\langle i \rangle\!\rangle_k$
$\mathsf{true}\ \mathsf{and}\ b \to b, \quad \mathsf{false}\ \mathsf{and}\ b \to \mathsf{false}$	

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. K *configurations* are specified using cells $\langle\!\langle\ S\ \rangle\!\rangle_h$, where h is a cell index and S is any term, in particular a multiset or a list of possibly other cells. $\mathsf{Set}[S]$ and $\mathsf{List}[S]$ 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 1 correspond to ($k, k_1, k_2 \in K$, $r_1 \in \mathit{KResult}$, $x \in \mathit{Name}$):

$k_1 + k_2 = k_1 \curvearrowright \square + k_2$	$k_1 \ \mathsf{and}\ k_2 = k_1 \curvearrowright \square \ \mathsf{and}\ k_2$
$k_1 + k_2 = k_2 \curvearrowright k_1 + \square$	$x := k = k \curvearrowright x := \square$
$k_1 \leq k_2 = k_1 \curvearrowright \square \leq k_2$	$\mathsf{if}\ k\ \mathsf{then}\ k_1\ \mathsf{else}\ k_2 = k \curvearrowright \mathsf{if}\ \square\ \mathsf{then}\ k_1\ \mathsf{else}\ k_2$
$r_1 \leq k_2 = k_2 \curvearrowright r_1 \leq \square$	$\mathsf{halt}\ k = k \curvearrowright \mathsf{halt}\ \square$
$\mathsf{not}\ k = k \curvearrowright \mathsf{not}\ \square$	

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

$$\langle\!|\underline{x := v}|\!\rangle_k \ \langle\!|(x, \underset{v}{\underline{\_}})|\!\rangle_{state}$$

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 the assignment statement. We prefer to use the more conventional notation "$l \to r$" 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.

$$\mathit{Stmt} ::= \ldots \mid \mathsf{spawn}\ \mathit{Stmt}$$

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:

$$\langle\!|\ \underset{\cdot}{\underline{\mathsf{spawn}(s)}}\ |\!\rangle_k \quad \frac{\cdot}{\langle\!|\ s\ |\!\rangle_k}$$

Therefore, multiple $(\!|\ \_\ |\!)_k$ 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 multi-threading, 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:

$$(\!|\ \cdot\ |\!)_k = \cdot$$

Synchronization mechanisms through lock acquire and release, as well as through rendez-vous 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[2]), 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.

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

[2] See [8], for a similar structure.

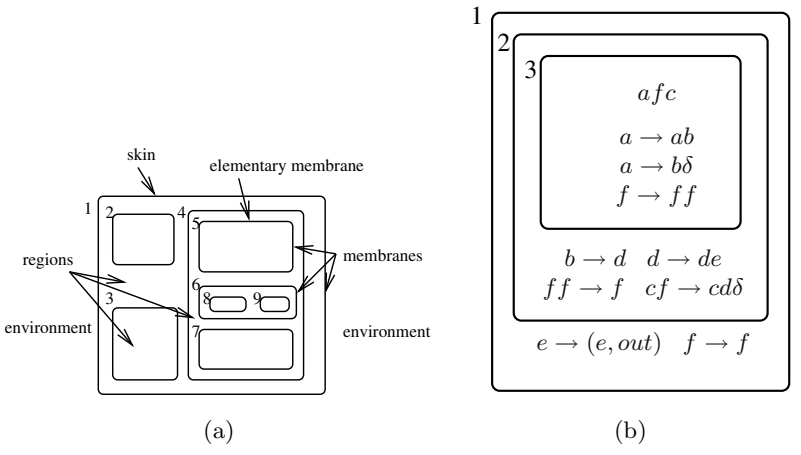


Fig. 1. A membrane system (a) and a "classical" P system (b)

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].

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

$u \to \overline{v}$ or $u \to \overline{v}\delta$
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; alternatively, i_o may be 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. 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 the region of membrane 3 (the other regions have no objects), then membrane 3 is dissolved and the evolution continues in the region of membrane 2, and when membrane 2 is dissolved, the final stage of the execution is in the region of membrane 1 (the skin membrane). In the region of 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 the region of membrane 2. In the region of 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 the region of 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 with a contents $[_h S]_h$ (S is the full contents of $[_h \ ]_h$) is represented in K as[3]

$$\langle\!| \ S \ |\!\rangle_h$$

The top configuration is $\langle\!| \ \langle\!| \ \ |\!\rangle_{skin} \ \ \langle\!| \ \ |\!\rangle_{env} \ |\!\rangle_\top$, including a representation of the objects in the environment.

[3] 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.

– The rules in Π are represented as global rules in $K(\Pi)$, their localization being a side-effect of membrane name matching. The evolution rules reduce to two basic rules used in a membrane $[_h\ ]_h$ 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 region of membrane $[_h\ ]_h$, v_i those that enter into other regions through internal membranes, and v_o those that go out through the membrane. The associated rule in K is the following: for any $k \geq 1$ and $v_i = v_1 \dots v_k$ (with nonempty v_j's) we have a rule

$$\langle\!\vert \ \underline{u}_{v_h}\ \langle\!\vert \underline{\cdot}_{v_1} \vert\!\rangle_{\_} \dots \langle\!\vert \underline{\cdot}_{v_k} \vert\!\rangle_{\_}\ \vert\!\rangle_h\ \underline{\cdot}_{v_o}$$

  - $u \to u\delta$ (δ indicates that the membrane is dissolved)

$$\langle\!\langle\ u\ z\ \rangle\!\rangle_h \to u\ z$$

For the first rule $u \to \overline{v}$, a second possibility would be to perform two steps: move v_h, v_o first, then for all remaining tagged objects (v_r, in) use a matching rule $\langle\!\vert\ (v_r, in)\ \langle\!\vert\ \underline{\cdot}_{v_r}\ \vert\!\rangle_{\_}\ \vert\!\rangle_{\_}$.

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, may be represented (in the simple case when all objects which enter through an internal membrane use the same membrane i) by

$$\langle\!\langle\ u\ \langle\!\langle w \rangle\!\rangle_i\ z\ \rangle\!\rangle_h \to v_h\ \langle\!\langle w\ v_i \rangle\!\rangle_i\ z\ v_o$$

In this interpretation, first the objects are moved out through 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

$$\langle\!\vert\ \underline{\langle\!\langle\ u\ \langle\!\vert\ w\ \vert\!\rangle_i\ z\ \rangle\!\rangle_h}_{v_h\ \langle\!\vert\ w v_i\ \vert\!\rangle_i\ z}\ \vert\!\rangle_{\_}\ \underline{\cdot}_{v_o}$$

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 a working hypothesis to keep the evolution simpler and to find results in the theoretical model.

4.3 Variations of Transition P Systems

Object movement. We start with a few variations of the object movement rule (presented in [22]), then we describe their associated rules in K.

– (deterministic *in*) In this variant, in_j is used instead of a simple *in* to indicate that an object goes into the region of the internal membrane j; its K-translation is

$$\langle\!\langle \underset{v_h}{u} \quad \langle\!\langle \underset{v_1}{\cdot} \rangle\!\rangle_{j_1} \ldots \langle\!\langle \underset{v_k}{\cdot} \rangle\!\rangle_{j_k} \rangle\!\rangle_h \quad \underset{v_o}{\cdot}$$

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 regions of 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. The objects with $+/-$ polarities go into the internal regions of 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. Therefore, it is easy to provide K definitions for these situations. 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 from a region to another region.

The P systems rule is $[_h u]_h \rightarrow [_{h'} v]_{h'}$. To represent this rule in K we need an explicit rule for matching membranes at different levels, denoted $\langle\!\langle^*\ \ ^*\rangle\!\rangle$ ($\langle\!\langle^*\ \ ^*\rangle\!\rangle$ 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

$$\langle\!\langle\ \langle\!\langle^*\ \underset{\cdot}{u}\ ^*\rangle\!\rangle_h \quad \langle\!\langle^*\ \underset{v}{\cdot}\ ^*\rangle\!\rangle_{h'}\ \rangle\!\rangle$$

- if the jump is into the region of an enclosed membrane, then

$$\langle\!\langle\ \underset{\cdot}{u} \quad \langle\!\langle^*\ \underset{v}{\cdot}\ ^*\rangle\!\rangle_{h'}\ \rangle\!\rangle_h$$

- if the jump is into the region of an outside membrane, then

$$\langle\!\langle\ \underset{v}{\cdot} \quad \langle\!\langle^*\ \underset{\cdot}{u}\ ^*\rangle\!\rangle_h\ \rangle\!\rangle_{h'}$$

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.

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 $l \to r$ 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):

$$\langle\!| \; \underset{r}{\underline{l}} \; z \; |\!\rangle_h$$

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

$$\langle\!| \; \underset{r}{\underline{l}} \; z \; |\!\rangle \quad \text{where } z \not\subseteq x$$

Complex side condition like the above are handled by means of conventional conditional rewrite rules in our Maude implementation of K.

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 $a \to b$ to take place; or (2) they may be seen as a way to control the parallelism, by restricting a free reaction $a \to b$ to the number of occurrences of the catalyst. In other extensions, 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 representing their behavior in K is straightforward. However, notice that c is required for a to become b in a catalyst $\langle \underset{bc}{\underline{ac}} \rangle_h$, while for a promoter a becomes b if c is present $\langle \underset{b}{\underline{a}} \; c \rangle_h$. In RWL, the two rules above would be identical, while in K, c actively participates to first rule, passively to the second (in the latter case, c may be shared by more rules applied in parallel).

Border rules. Border rules are particular object evolution rules of the following type $xu[_i vy \rightarrow xu'[_i v'y$. They allow to test and modify the objects from two neighboring regions. Such a rule may be represented in K by

$$x \ \frac{u}{u'} \ \langle\!\{ \ \frac{v}{v'} \ y \ \}\!\rangle_{\_}$$

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 $1, \ldots, m$:

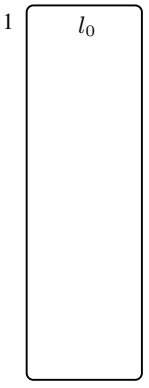
- a symport rule is of the form (x, in) or (x, out), where $x \in V^+$, with the meaning that the objects specified by x enter, respectively exit, the membrane, and
- an antiport rule is of the form $(x, in; y, out)$, where $x, y \in V^+$, which means that x is taken into the membrane region from the surrounding region and 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.[4] 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.

[4] 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 $l_h : \texttt{halt}$:
- (l_h, out)

Fig. 2. A "symport/antiport" P system

Example. An example is presented in Fig. 2. It describes how symport/antiport P systems may simulate counter machines (it is well known that counter machines 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. 2, we use an equivalent definition of counter machines where the statements are 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)

The simulation works as follows: The statement to be executed next 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

$$\langle\!| \ x \ |\!\rangle_{env} = \langle\!| \ x \ |\!\rangle_{env} \ x$$

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.

6.1 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 other 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.

6.2 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: $[_h a \to v]^e_h$ $(h \in H, e \in \{+, -, 0\}, a \in O, v \in O^*)$

$$\langle\!| \; \underset{v}{\underline{a}} \; |\!\rangle^e_h$$

– "in" communication rules: $a[_h]^{e_1}_h \to [_h b]^{e_2}_h$ $(h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O)$

$$\underset{\cdot}{\underline{a}} \quad \langle\!| \; \underset{b}{\underline{\cdot}} \; |\!\rangle^{\frac{e_1}{e_2}}_h$$

– "out" communication rules: $[_h a]^{e_1}_h \to [_h]^{e_2}_h b$ $(h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O)$

$$\langle\!| \; \underset{\cdot}{\underline{a}} \; |\!\rangle^{\frac{e_1}{e_2}}_h \quad \underset{b}{\underline{\cdot}}$$

– dissolving rules: $[_h a]^e_h \to b$ $(h \in H, e \in \{+, -, 0\}, a, b \in O)$

$$(\!| \; a \quad x \; |\!)^e_h \to b \quad x$$

– division rules for elementary membranes: $[_h a]^{e_1}_h \to [_h b]^{e_2}_h [_h c]^{e_3}_h$ $(h \in H, e1, e2, e3 \in \{+, -, 0\}, a, b, c \in O)$

$$(\!| \; a \quad x \; |\!)^{e_1}_h \to (\!| \; b \quad x \; |\!)^{e_2}_h \quad (\!| \; c \quad x \; |\!)^{e_3}_h$$

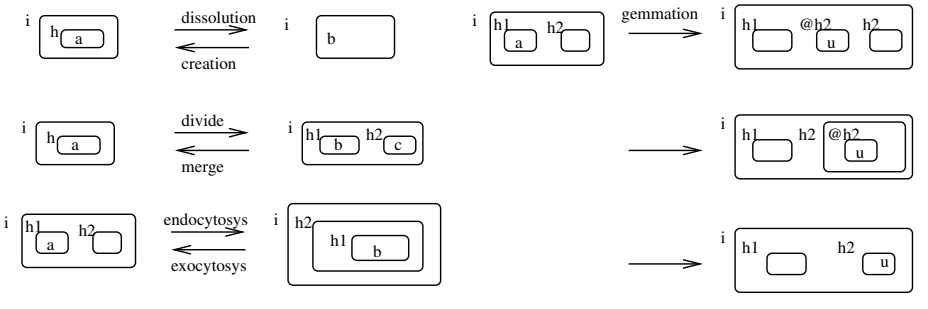


Fig. 3. Membrane handling operations

6.3 Variations of P Systems with Mctive 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

$$a \to \langle\!\langle\ v\ \rangle\!\rangle_h$$

Merging of membranes. This rule is $[_h x]_h[_{h'} y]_{h'} \to [_{h''} z]_{h''}$, allowing to merge the contents of two neighboring membranes (the rules for $[_{h''}..]_{h''}$ are taken from a dictionary). The K rule is

$$\langle\!\langle\ x\ \rangle\!\rangle_h \quad \langle\!\langle\ y\ \rangle\!\rangle_{h'} \to \langle\!\langle\ z\ \rangle\!\rangle_{h''}$$

Split of membranes. This operation is opposite to merge. Its format is $[_{h''} z]_{h''} \to [_h x]_h[_{h'} y]_{h'}$ and the K rule is

$$\langle\!\langle\ z\ \rangle\!\rangle_{h''} \to \langle\!\langle\ x\ \rangle\!\rangle_h \quad \langle\!\langle\ y\ \rangle\!\rangle_{h'}$$

One appealing version is to put into a membrane the objects of a given type and in the other the remaining ones.

Endocytosis and exocytosis. Endocytosis is a rule $[_h x]_h[_{h'}]_{h'} \to [_{h'}[_h y]_h]_{h'}$, i.e., in one step a membrane and its contents enter into a neighboring membrane. The K rule is

$$\frac{\langle\!\langle\ x\ \rangle\!\rangle_h}{\cdot} \quad \langle\!\langle\ \frac{\cdot}{\langle\!\langle\ y\ \rangle\!\rangle_h}\ \rangle\!\rangle_{h'}$$

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.[5] 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.[6] The illustrating example of computing n^2 with the P system in Fig. 1(b) is not a fortunate one. For instance,

[5] One does not consider the complicated case when the delivery gets lost into the system due to a reconfiguration of the membrane structure.

[6] A few P systems with particular structured objects (strings, conformons) are presented in [20,12].

Table 3. Runs for P systems with and without data structure on objects

(a)

square	time(ms)	rewritings	parallel steps
6	11	2316	13
10	123	20012	21
15	3882	562055	31
18	32294	4463244	37
19	failure		

(b)

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		

(c)

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		

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 more than the atoms in the Universe.[7] 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.

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. 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 3(a).

[7] The Universe is estimated to 4×10^{79} hydrogen atoms, while 2^{999} is roughly 10^{300}.

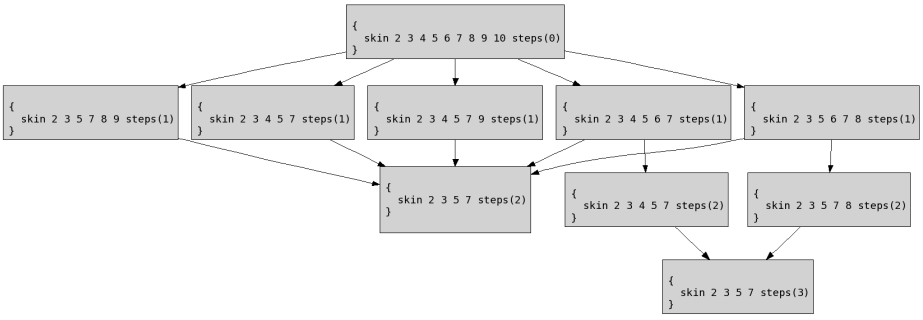


Fig. 4. The graph of computing the prime numbers up to 10

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 3(b),(c). The transition graph for the example with prime numbers up to $n = 10$ is displayed in Fig. 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.

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 [2,3,4,5]. 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 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.

References

1. Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., Watson, J.D.: Molecular Biology of the Cell, 3rd edn. Garland Publishing, New York (1994)
2. Andrei, O., Ciobanu, G., Lucanu, D.: Structural operational semantics of P systems. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 31–48. Springer, Heidelberg (2006)
3. Andrei, O., Ciobanu, G., Lucanu, D.: Expressing control mechanisms of membranes by rewriting strategies. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 154–169. Springer, Heidelberg (2006)
4. Andrei, O., Ciobanu, G., Lucanu, D.: Operational semantics and rewriting logic in membrane computing. Electr. Notes Theor. Comput. Sci. 156, 57–78 (2006)
5. Andrei, O., Ciobanu, G., Lucanu, D.: A rewriting logic framework for operational semantics of membrane systems. Theoretical Computer Sci. 373, 163–181 (2007)
6. Banatre, J.-P., Coutant, A., Le Metayer, D.: A parallel machine for multiset transformation and its programming style. Future Generation Computer Systems 4, 133–144 (1988)
7. Berry, G., Boudol, G.: The chemical abstract machine. Theoretical Computer Sci. 96, 217–248 (1992)
8. Cardelli, L.: Brane calculi. In: Danos, V., Schachter, V. (eds.) CMSB 2004. LNCS (LNBI), vol. 3082, pp. 257–278. Springer, Heidelberg (2005)
9. Ciobanu, G., Păun, G., Stefănescu, G.: P transducers. New Generation Computing 24, 1–28 (2006)
10. Clavel, M., Duran, F., Eker, S., Lincoln, P., Marti-Oliet, N., Meseguer, J., Quesada, J.F.: Maude: specification and programming in rewriting logic. Theoretical Computer Sci. 285, 187–243 (2002)
11. Davis, M., Sigal, R., Weyuker, E.J.: Computability, Complexity, and Languages, 2nd edn. Morgan Kaufmann, San Francisco (1994)
12. Frisco, P.: The conformon-P system: A molecular and cell biology-inspired computability model. Theoretical Computer Sci. 312, 295–319 (2004)
13. Hills, M., Rosu, G.: KOOL: An application of rewriting logic to language prototyping and analysis. In: Baader, F. (ed.) RTA 2007. LNCS, vol. 4533, pp. 246–256. Springer, Heidelberg (2007)
14. Hills, M., Serbanuta, T., Rosu, G.: A rewrite framework for language definitions and for generation of efficient interpreters. Electr. Notes Theor. Comput. Sci. 176, 215–231 (2007)
15. Meseguer, J.: Conditioned rewriting logic as a united model of concurrency. Theoretical Computer Sci. 96, 73–155 (1992)
16. Meredith, P., Hills, M., Rosu, G.: A K Definition of Scheme. Technical report UIUCDCS-R-2007-2907 (October 2007)
17. Milner, R.: A theory of type polymorphism in programming. J. Computer System Sci. 17, 348–375 (1978)

18. Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)
19. Păun, A., Păun, G.: The power of communication: P systems with symport/antiport. New Generation Computing 20, 295–306 (2002)
20. Păun, G.: Computing with membranes. J. Computer and System Sci. 61, 108–143 (2000)
21. Păun, G.: P systems with active membranes: Attacking NP-complete problems. J. Automata, Languages and Combinatorics 6, 75–90 (2001)
22. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
23. Rosu, G.: K: A rewriting-based framework for computations – Preliminary version. Technical Report UIUCDCS-R-2007-2926, Department of Computer Science, University of Illinois. 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 (2007), `http://fsl.cs.uiuc.edu/k`
24. The Web Page of Membrane Computing, `http://ppage.psystems.eu/`

Translating Multiset Tree Automata into P Systems*

José M. Sempere

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
jsempere@dsic.upv.es

Abstract. In this work we propose a translation scheme to obtain P systems with membrane creation 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. For instance, in [8] we introduced multiset tree automata, and in [5] this model was used to calculate editing distances between membrane structures. A method to infer multiset tree automata from membrane observations was presented in [9], while in [10] 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 creation rules from a set of trees that model the desired behavior of the P system structure according to [9].

2 Notation and Definitions

In the sequel we provide some concepts from formal language theory, membrane systems and multiset processing. We suggest the books [7], [6] and [1] to the reader.

Multisets

First, we provide some definitions from multiset theory as exposed in [11].

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 $A \oplus B$.

* Work supported by the Spanish Ministerio de Ciencia e Innovación under project TIN2007-60769.

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 394–402, 2009.

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 $\Psi : D^* \to \mathbb{N}^n$ 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 [6]. A transition P system of degree m is a construct

$$\Pi = (V, T, C, \mu, w_1, \cdots, w_m, (R_1, \rho_1), \cdots, (R_m, \rho_m), i_0), \text{ where:}$$

- 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 $u \to v$) where u is a string over V and $v = v'$ or $v = v'\delta$ where v' is a string over

  $$\{a_{here}, a_{out}, a_{in_j} \mid a \in V, 1 \leq j \leq m\}$$

  and δ is a symbol not in V that defines the *membrane dissolving action*. From now on, we denote the set *tar* by $\{here, out, in_k \mid 1 \leq k \leq m\}$,
- 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 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 investigated or they have been employed for different purposes.

In the following, we enumerate two kinds of rules which are able to modify the membrane structure.

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.
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 reached by Π during its computation. Observe that this definition was used in [3], in order to define tree languages generated by Π systems. In such case, only the membrane structures obtained after halting were considered.

In this work we introduce a new kind of rules which allow the creation of different regions in only one step: n-creation. Formally, a n-creation rule is defined as $a \rightarrow w_0 [_{h_1} w_1]_{h_1} [_{h_2} w_2]_{h_2} \cdots [_{h_n} w_n]_{h_n}$. The meaning of this rule is that the object a is transformed into objects $w_0, w_1, \cdots, w_n$. Then, n new regions are created with (probably repeated) labels $h_1, h_2, \cdots, h_n$, and the new objects are placed in the new regions. This rule is a generalization of the creation rule proposed in different works; we refer to [6] for references. Observe that, under our point of view, the use of a n-creation rule is equivalent to the use of a creation rule together with a division rule and then a dissolving rule in order to erase the external region.

Tree Automata and Tree Languages

Now, we introduce some concepts from tree languages and automata as exposed in [2,4]. 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, $perm_1(t) = \{\sigma(t_{i_1}, ..., t_{i_n}) \mid \langle i_1, i_2, ..., i_n\rangle \in perm(\langle 1, 2, ..., n\rangle)\}$.

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

$$Ant(q) = \{p_1 \cdots p_n \mid p_i \in Q \cup V_0 \wedge \delta_n(\sigma, p_1, ..., p_n) = q\}$$

From now on, we refer to deterministic finite tree automata simply as *tree automata*. We suggest [2,4] for other definitions on tree automata.

The transition function δ is extended to a function $\delta : V^T \to Q \cup V_0$ on trees as follows:

$$\delta(a) = a \text{ for any } a \in V_0$$
$$\delta(t) = \delta_n(\sigma, \delta(t_1), \ldots, \delta(t_n)) \text{ for } t = \sigma(t_1, \ldots, t_n) \ (n > 0)$$

Note that the symbol δ denotes both the set of transition functions of the automaton and the extension of these functions to operate on trees. In addition, one can observe that the 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:*

$$\delta = \bigcup_{\substack{1 \leq i \leq n \\ i : V_i \neq \emptyset}} \delta_i$$

$$\begin{aligned} \delta_i : (V_i \times \mathcal{M}_i(Q \cup V_0)) &\rightarrow \mathcal{P}(\mathcal{M}_1(Q)) \qquad i = 1, \ldots, n \\ \delta_0(a) = M_\Psi(a) &\in \mathcal{M}_1(Q \cup V_0) \qquad \forall a \in V_0 \end{aligned}$$

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 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, [8]) 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, [8]) 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 [8]. 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, [8]) 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, [8]) 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:

$$\delta(\sigma, aa) = q_1,\ \delta(\sigma, a) = q_2,\ \delta(\sigma, aq_2) = q_2,\ \delta(\sigma, q_1q_1) = q_1,\ \delta(\sigma, aq_2q_1) = q_3 \in F.$$

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
$R_{q_3} = \{b \to a[_{q_2}b]_{q_2}[_{q_1}b]_{q_1}\}$,
$R_{q_2} = \{b \to a[_{q_2}b]_{q_2}; b \to a\}$, 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.

In the following example the number of final states is greater than one.

Example 2. Let us take the MTA defined by the following transitions

$$t_1 : \delta(\sigma, aa) = q_1,\ t_2 : \delta(\sigma, bb) = q_2,\ t_3 : \delta(\sigma, q_2q_2) = q_2,\ t_4 : \delta(\sigma, q_1q_1) = q_1,\ t_5 : \delta(\sigma, q_2q_1) = q_3.$$

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
$R_0 = \{c \to aa^{(1)}; c \to bb^{(2)}; c \to [_{q_2}c]_{q_2}[_{q_2}c]^{(3)}_{q_2}; c \to [_{q_1}c]_{q_1}[_{q_1}c]^{(4)}_{q_1};$
$c \to [_{q_2}c]_{q_2}[_{q_1}c]^{(5)}_{q_1}\}$
$R_{q_2} = \{c \to bb^{(2)}; c \to [_{q_2}c]_{q_2}[_{q_2}c]^{(3)}_{q_2}\}$
$R_{q_1} = \{c \to aa^{(1)}; c \to [_{q_1}c]_{q_1}[_{q_1}c]^{(4)}_{q_1}\}$

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.

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

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.

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 n-creation rules (or membrane creation plus membrane division) and the trees accepted by MTA. This result was pointed out in previous works such as [5,8,9,10].

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 [9]. 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 [10]. 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. The author is grateful to the reviewers for sharp remarks and suggestions made to this work. The author is most indebted to Mario J. Pérez-Jiménez and Gheorghe Păun for comments on n-creation rules during the *9th Workshop on Membrane Computing* at Edinburgh.

References

1. Calude, C.S., Păun, G., Rozenberg, G., Salomaa, A. (eds.): Multiset Processing. LNCS, vol. 2235. Springer, Heidelberg (2001)
2. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (1997) (October 1st, 2002), `http://www.grappa.univ-lille3.fr/tata`
3. Freund, R., Oswald, M., Păun, A.: P systems generating trees. In: Mauri, G., Păun, G., Jesús Pérez-Jímenez, M., Rozenberg, G., Salomaa, A. (eds.) WMC 2004. LNCS, vol. 3365, pp. 309–319. Springer, Heidelberg (2005)
4. Gécseg, F., Steinby, M.: Tree languages. In: Handbook of Formal Languages, vol. 3, pp. 1–69. Springer, Heidelberg (1997)
5. López, D., Sempere, J.M.: Editing distances between membrane structures. In: Freund, R., et al. (eds.) WMC 2005. LNCS, vol. 3850, pp. 326–341. Springer, Heidelberg (2006)
6. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
7. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer, Heidelberg (1997)
8. Sempere, J.M., López, D.: Recognizing membrane structures with tree automata. In: Gutiérrez Naranjo, M.A., et al. (eds.) Proc. 3rd Brainstorming Week on Membrane Computing, Fénix Editora, Sevilla, pp. 305–316 (2005)
9. Sempere, J.M., López, D.: Identifying P rules from membrane structures with an error-correcting approach. In: Hoogeboom, H.J., et al. (eds.) WMC 2006. LNCS, vol. 4361, pp. 507–520. Springer, Heidelberg (2006)
10. Sempere, J.M., López, D.: Characterizing membrane structures through multiset tree automata. In: Eleftherakis, G., et al. (eds.) WMC 2007. LNCS, vol. 4860, pp. 428–437. Springer, Heidelberg (2007)
11. Syropoulos, A.: Mathematics of multisets. In: [1], pp. 347–358

Author Index