Chapter 1
Modelling and analysis of *E. coli* respiratory chain

Adrian Ţurcanu[1], Laurenţiu Mierlă[1], Florentin Ipate[1,2], Alin Stefanescu[1], Hao Bai[3], Mike Holcombe[3], Simon Coakley[4]

Abstract In this chapter we present some results obtained in the study of the bacterium *E. coli* related to its behavior at different level of oxygen in the environment. The biological model is expressed in terms of different molecules and their reactions. First, an agent-based model of *E. coli* is implemented in the FLAME framework for multi-agents and some simulation results are given. Each agent is represented by an X-machine and the model corresponds to communicating X-machines. Then this model is transformed into a kernel P system. This kernel P system is implemented in the Rodin platform and in Spin and some properties are verified using the associated model checkers. Formulated using the LTL formalism, the verified properties refer to the variation of the number of different molecules as a result of the occurring reactions. Our main contribution is a simplified model of *E. coli* that preserves the main properties of the initial model, and can be formally verified using a model checker.

1.1 Introduction

Membrane computing, a research field introduced in 1998 by Gheorghe Păun, studies computing devices inspired by the functioning and structure of the living cell, called P systems [33]. Since the appearance of membrane computing, many variants of P systems have been defined and investigated, particularly in terms of computational power and their capability to solve computationally hard problems [34].

[1] University of Piteşti, Department of Computer Science, Romania
e-mail:`adrianturcanu85@yahoo.com, laurentiu.mierla@gmail.com, alin@stefanescu.eu`
[2] University of Bucharest, Department of Computer Science, Romania
e-mail:`florentin.ipate@ifsoft.ro`
[3] University of Sheffield, Department of Computer Science, UK
e-mail:`{hao.bai, s.coakley}@sheffield.ac.uk, m.holcombe@epigenesys.co.uk`

Introduced in [18], kernel P systems (kP systems), and its reduced variant [20] (skP systems) represent an unifying framework for P systems which integrates many features of existing P systems into an elegant and yet powerful modelling formalism.

In the last years, significant developments have also been made in using P systems to model, simulate and formally verify various systems [13]. As a consequence, the idea of automating the evolution of a P system was one of the main concerns of the membrane computing community and many tools have been developed, as surveyed in [15]. However, each of these tools came with its own specification language, and was dedicated to a certain class of P systems. In this context, the Research Group of Natural Computing, from the University of Seville, developed P-Lingua [16], a programming language for P systems that became the standard for their representation. Another useful tool developed by the same research group is MeCoSim [32], a membrane computing simulator that can be easily adapted to each family of P systems. Using a P-Lingua definition file of a P system, MeCoSim can be used for simulations and property extraction.

Formal verification of biological systems has been studied using e.g., rewriting logic and the Maude tool [9] or PRISM and the associated probabilistic temporal logic [21] for stochastic systems [11]. More recently, various properties of transition P systems, P systems with active membranes and kernel P systems have been verified using different tools like NUSMV [19], SPIN [27] and RODIN [28, 36].

Based on the X-machine formalism [22], FLAME (Flexible Large-scale Agent Modelling Environment) [1, 35] is a generic agent-based modelling system, which can be used to develop and simulate applications in many areas. In particular, the FLAME framework has been proven very successful for modelling and simulating different biological, economic and social systems. In FLAME, each agent has a memory that holds variables and evolves according to a transition diagram in which the transitions are labelled by processing functions. These functions can read and write to variables in the agent's memory or can read incoming messages and write outgoing messages. The agents communicate via messages. One of the great strengths of FLAME is its modularization and platform independence, which allows it to be run on parallel supercomputers.

One of the success stories of FLAME is the SUMO (Systems Understanding of Microbial Oxygen Responses) research project [5], funded by the European research initiative SysMO (System Biology of Microorganisms), centered around the *Escherichia coli (E. coli)* bacterium, one of the most studied organism in biology [23]. In particular, the project investigates the behavior of this bacterium related to its reaction to the level of oxygen in the environment. Using the mathematical models and simulation results provided by FLAME, SUMO deepened the existing knowledge about the metabolic adaption that occurs in response to changes in oxygen availability [5].

A number of recent investigations [20, 26] illustrate the expressive power of kP systems on several case studies, representing known NP-complete problems. This chapter makes a further step in this direction, by using the kP system as a modelling tool for biological systems. To this end, we show how the X-machine based models of the *E. coli* developed in FLAME can be naturally transformed into kP system models. Furthermore, these kP systems also provide the basis for the implementation in two modelling languages (Event-B and Promela) and the associated model checkers, ProB [3,30] and Spin [4,24], are used to simulate and verify some of their properties. In related papers, the probabilistic model checking tool PRISM was used for formal verification in systems biology for quantitative properties [29] or stochastic trend formulae [8]. Introduced in [12], the Infobiotics Workbench is also a powerful computational framework incorporating model specification, simulation, parameter optimisation and model checking for various systems biology problems. Finally, BMA tool [10] makes formal methods accessible to biologists by means of an intuitive visual interface.

The chapter is structured as follows. In the next section we overview the theoretical models of kP systems and X-machines. We continue by describing *E. coli* and its respiratory chain in Section 1.3 and its FLAME simulation in Section 1.4. Section 1.5 shows how to use the kP systems to capture interesting and relevant aspects of *E. coli* behaviour. The verification of different properties is implemented using two different model-checkers in Section 1.6. Conclusions are drawn in the last section.

1.2 Background

In this section we provide the formal definitions of a variant of P systems and of X-machines.

1.2.1 kP systems

P systems are distributed and parallel computing devices processing multisets of objects encapsulated into regions delimited by membranes, using various types of rules (evolution, communication, division and others). Kernel P systems provide an unifying framework for many features available in various P system variants [18]. In this chapter we will use a simplified version of kernel P systems called simple kernel P systems [20].

Definition 1. A *simple kernel P system* (skP system, for short) of degree $n \geq 1$ is a tuple

$$sk\Pi = (A, L, IO, C_1, \ldots, C_n, \mu, i_0)$$

where :

- A is an alphabet containing objects,
- L is a finite set of labels,
- IO is an alphabet, $IO \subset A$, associated with the environment,
- $C_1, \ldots, C_n$ are the initial compartments of the system; each of them is identified by a label of L, has initially a multiset over A, and a finite set of rules,

- $\mu = (V, E)$ is an undirected graph, where $V \subseteq L$ are vertices and E the edges, and
- $i_0 \in L \cup \{0\}$ denotes the *output region*, i.e., the compartment receiving the result of a computation.

A skP system $(A, L, IO, C_1, \ldots, C_n, \mu, i_0)$ can be viewed as a set of n compartments $C_1, \ldots, C_n$, interconnected by edges from E of an undirected graph μ. Every compartment has an associated set of rules that can be of type division, rewriting or communication. Rules may have guards (necessary conditions) and are applied in maximally parallel mode. The only restrictions are that at most one division rule can be applied per step and, when a cell is divided, the division rule is the only one which is applied for that cell in that step. We describe the guards and the rules syntax below.

The guards are constructed using multisets over A, relational and Boolean operators. Before defining it, we introduce some notations. For a multiset w over A and an element $a \in A$, we denote by $|w|_a$ the number of a's occurring in w. Let $Rel = \{<, \leq, =, \neq, \geq, >\}$ be the set of relational operators, $\gamma \in Rel$ a relational operator, a^n a multiset and $r\{g\}$ a rule with guard g.

Definition 2. If g is the abstract relational expression γa^n and the current multiset is w, then the guard is true for the multiset w if $|w|_a \gamma n$ is true.

Abstract relational expressions can be connected by Boolean operators ($\neg$, $\wedge$ and $\vee$) generating *abstract Boolean expressions*.

Definition 3. If g is the abstract Boolean expression and the current multiset is w, then the guard denotes the Boolean expression for w, obtained by replacing abstract relational expressions with relational expressions for w. The guard g is true for the multiset w when the Boolean expression for w is true.

Definition 4. A guard is defined recursively as:
(i) one of the Boolean constants $true$ or $false$, or
(ii) an abstract relational expression, or
(iii) an abstract boolean expression.

Example. The guard $g = \geq a^3 \wedge \geq b^4 \vee \neg > c$ is true for w if w contains at least 3 a's and 4 b's or no more than one c.

The rules can have one the following syntax:

(a) rewriting and communication rule: $x \to y \ \{g\}$,
where $x \in A^+$, $y \in A^*$. The right hand side y, has the form $y = (a_1, t_1) \ldots (a_h, t_h)$, where $a_j \in A$ and $t_j \in L$, $1 \leq j \leq h$, is an object and a target, i.e., the label of a compartment, respectively, and $(a_i, t_i) \neq (a_j, t_j)$, for each $1 \leq i, j \leq h$, $i \neq j$. The target t_j, must be either the label of the current compartment, say l_i, (most often ignored) or that of an existing neighbour of it ($(l_i, t_j) \in E$) or an unspecified one, $*$; otherwise, the rule is not applicable. If a target t_j refers to a label that appears more than once, then one of the involved compartments will be non-deterministically chosen. If t_j is $*$ then the object a_j is sent to a neighbouring compartment arbitrarily chosen.

Example. If the rule is $r : a \to a(b,2)(c,3)\ \{g\}$, then it is applicable iff the guard g is true, and, as a result of its application, one object a stays in the current compartment (we do not use target for it), one object b is sent to the compartment labelled 2 and one object c is sent to the compartment labelled 3.

(b) membrane division rule: $[x]_{l_i} \to [y_1]_{l_{i_1}} \dots [y_h]_{l_{i_h}}\ \{g\}$, where $x \in A^+$ and $y_j = (a_{j,1},t_{j,1})\dots(a_{j,h_j},t_{j,h_j})$, where $a_{j,k} \in A$, $t_{j,k} \in L$, and $1 \leq k \leq h_j$. In this case, the compartment l_i will be replaced by h compartments, with the labels $l_{i_1},\dots,l_{i_h}$; furthermore, for $1 \leq j \leq h$, the i_j-th compartment will contain the same objects as l_i with the exception of x, which will be replaced by y_j. Moreover, all the links of l_i are inherited by each of the newly created compartments.
Example. If the rule is $r : []_2 \to []_{21}[]_{22}[]_{23}\ \{g\}$, then it is applicable iff the guard g is true, and, as a result, the compartment with label 2 is replaced with 3 compartments with the same content as compartment 2.

In our models only rewriting and communication rules will be used. Furthermore, all such rules are non-cooperative, i.e., only one object appears on the left side of each rule.

Since we are dealing with the model of a biological system, probabilities are added to the rules. The idea of adding probabilities to P systems was based on the intention to keep membrane system theory as close as possible to the biological reality [31].

We consider that probabilistic rules are complementary, i.e., these can be grouped into sets (pairs in our case) with the same left side and the sum of the probabilities associated with the rules of each pair being 100%. Thus, if $r_1 : x \xrightarrow{p\%} y$ and $r_2 : x \xrightarrow{(100-p)\%} z$ are two probabilistic rules, then rule r_1 is applied with a probability of $p\%$ and rule r_2 with a probability of $(100-p)\%$. After being associated with the objects according to their probabilities, rules are applied in a maximal parallel manner in each compartment. (The difference between the original, non-deterministic, kP system and the probabilistic one is that, at each moment within a maximally parallel computation step, whenever more than one rule can be selected, the applied rule is selected according to its probability rather than non-deterministically.)

1.2.2 X-machines

Introduced in 1974 by Samuel Eilenberg [17], X-machines were proposed as a basis for a possible specification language by Mike Holcombe in 1988 [22]. They provide the modelling foundation for the FLAME framework. X-machines are computational models that can describe a system as a finite set of states, each with an internal store called memory, and a number of transitions between the states. A transition is triggered by an input value, produces an output value and may alter the memory. An X-machine may be modelled by a finite automaton in which the arcs are labelled by function names (the *processing functions*).

Definition 5. An *X-Machine* is a tuple

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

where:

- Σ and Γ are finite sets called *input alphabet* and respectively *output alphabet*,
- Q is the finite set of *states*,
- M is a (possibly) infinite set called *memory*,
- Φ is the *type* of XM, a non-empty finite set of *function symbols*. A *basic processing function* $\phi : M \times \Sigma \longrightarrow \Gamma \times M$ is associated with each function symbol ϕ.
- F is the (partial) *next state function*, $F : Q \times \Phi \longrightarrow 2^Q$,
  As for finite automata, F is usually described by a *state-transition diagram*.
- I and T are the sets of initial and terminal states respectively, $I \subseteq Q, T \subseteq Q$, and
- $m_0 \in M$ is the initial memory value.

Definition 6. A *communicating X-machine system* [25] with n components is a tuple $S_n = ((XM_i)_{1 \leq i \leq n}, R)$, where:

- $XM_i = (\Sigma_i, \Gamma_i, Q_i, M_i, \Phi_i, F_i, I_i, T_i, (m_0)_i)$ is an X-machine labelled by i, for $1 \leq i \leq n$, and
- R is a relation defining the communication among the components, $R \subseteq \{XM_1, XM_2, \ldots, XM_n\} \times \{XM_1, XM_2, \ldots, XM_n\}$. A tuple $(XM_i, XM_j) \in R$, denotes that the X-machine XM_i can output a message to a corresponding input stream of X-machine XM_j, for any $i, j \in \{1, \ldots, n\}$ with $i \neq j$.

The exchange of messages among the components of a communicating X-machine is achieved by redirecting one component's function output to be received as input by a function of another machine.

1.3 General description of *E. coli*

E. coli is one of the most studied bacterium and the research related to it provided many fundamental paradigms in biology. It can be easily handled by biologists and its genetic information and metabolic processes are well understood. Interestingly, unlike many organisms, *E. coli* can thrive in environments either with abundant oxygen or no oxygen.

A system-level study of the mechanism of *E. coli* responding to oxygen is a key to understand the respiratory pathways of this bacterium. Based on FLAME, an agent-based model was introduced to better understand the respiratory chain and to simulate the activities of relative components, such as oxidases *Cyo* and *Cyd*, and their regulators, *Fnr* and *ArcA*. In this model, the expression of *Cyd* and *Cyo* in *E. coli* are repressed or activated by *Fnr* and *ArcA*. An integration of COPASI (a software application for simulation and analysis of biochemical networks and their dynamics) makes it possible to calculate the dynamic variation of *Cyo*/*Cyd* numbers using mathematical methods. Compared with the traditional kinetic models, which consider the system as a macroscopic quantity, the agent-based model represents every single molecule and enables the activities of agents in an actual spatial region. This advantage of agent-based model could crucially complement the kinetic model, for the cases when the latter fails, such as low molecule number or unevenly distributed molecules.

In this agent-based model, each individual molecule of interest is defined as an agent with its own parameters, such as position, status etc. According to the biochemical reaction conditions, these agents could exist within the cellular environment and interact with each other. This cellular space can be defined as a 2-dimensional or 3-dimensional space, in which the molecules may be close to the membrane or evenly distributed in cytoplasm. All these molecules are capable of moving through this space and interacting with others according to their interaction radius. The current agent model consists of the following types of agents: oxygen (*O2* molecules), *E. coli* cell, *Fnr* molecules, *ArcB* molecules, *ArcA* molecules, *ArcBA* molecules, *Cyo* and *Cyd* molecules. This is a preliminary model, but we are currently working with a team of biologists on an improved version much closer to the real biological system.

The FLAME framework allows to define agents in a precise way and stores their initial specifications in an XML file. On reading this file, the pre-defined agents are constructed in a virtual space for further activities. Each agent communicates with the others via message boards. These messages contain information on the whereabouts and state of the molecule. Together with a random moving algorithm and pre-defined interaction rules, this information drives the whole model and leads to a final output. In the current model, the *Fnr* molecules are divided into three groups based on their status, including *Fnr* dimer, *Fnr* monomer and *Fnr* dimer bound to the binding sites on DNA. The mechanism of interaction between *Fnr* and oxygen molecule is defined as follows:

- When an oxygen molecule is within a pre-defined reaction distance to an *Fnr* dimer, the *Fnr* dimer is decomposed into two *Fnr* monomers. If this dimer is bound to a binding site, the binding site will become unoccupied.
- When two *Fnr* monomers are within reaction distance, they can be combined into an *Fnr* dimer.
- When the distance between a *Fnr* dimer and an unoccupied binding site is less than their reaction distance, the dimer will bind to the binding site.

The ArcBA molecules are present as *ArcA* octamer, *ArcA* tetramer, *ArcA* dimer(p), *ArcB* and *ArcA* octamer bound to the binding sites on DNAs. The *ArcA* octamers bound to binding sites have a probability of coming off. The *ArcB* molecules can be phosphorylated (*ArcB* P) or dephosphorylated (*ArcB*), depending on how much it was exposed to oxygen. The *ArcA* dimers can also be in these two forms. All the *ArcA* octamers and tetramers contain phosphorus. The mechanism of interaction between *ArcBA* molecules and oxygen is defined as:

- When an oxygen molecule is within pre-defined reaction distance to an *ArcB*, the *ArcB* will be de-phosphorylated and then become able to capture phosphorus from active *ArcA* dimers, *ArcA* tetramers and *ArcA* octamers. When an *ArcB* has no oxygen molecule to interact with, the capability of this *ArcB* of capturing phosphorus will reduce proportionally by revising its reaction radius.
- When an *ArcA* octamer is moving into the reaction distance to *ArcB*, the *ArcA* octamer will be de-phosphorylated and decomposed into one *ArcA* tetramer, one active *ArcA* dimer and one inactive *ArcA* dimer.
- When an *ArcA* tetramer is moving into the reaction distance to *ArcB*, the *ArcA* tetramer will be de-phosphorylated and decomposed into one active *ArcA* dimer and one inactive *ArcA* dimer.
- When an active *ArcA* dimer is moving into the reaction distance to *ArcB*, the *ArcA* dimer will be de-phosphorylated.
- When an inactive *ArcA* dimer is moving into the reaction distance to *ArcB* P, the *ArcA* dimer will be phosphorylated.
- When two active *ArcA* dimers are within their reaction distance, they can be combined into an *ArcA* tetramer.
- When two *ArcA* tetramers are within their reaction distance, they can be combined into an *ArcA* octamer.
- When one *ArcA* octamer is within its reaction distance to an available binding site, the *ArcA* octamer will bind to the binding site.

1.4 FLAME simulations of *E. coli* respiratory chain

As we stated before, the *E. coli* FLAME model is based on agents corresponding to the molecules. At every step in the simulation, each molecule must change its location (move) according to predefined rules. The size of the molecules is assumed to be sufficiently small that collision between them can be neglected in the movement process.

Each agent is represented by an X-machine whose memory contains an ID, the type of the agent, the physical location and its state. The behavior of each molecule is modelled in the rules of the corresponding agent: namely, with what molecules it can interact, and what distance is necessary for this interaction to occur. Agents communicate by sending and receiving messages containing values from their memory or announcing their availability to interact. Each agent contains a function that calculates the distance to all the other agents by using its coordinates and the coordinates received from other agents through messages. Besides the criteria on proximity (i.e., the calculated distance is less than the pre-defined interaction radius), two molecules interact if they are in a state that allows interactions. Each X-machine also contains functions that compute the total number of molecules of that type, move the molecules, send availability to other agents or destroy the agent if it is consumed in a reaction. These functions also change the state of the X-machine accordingly. Thus, the model corresponds to a communicating X-machine.

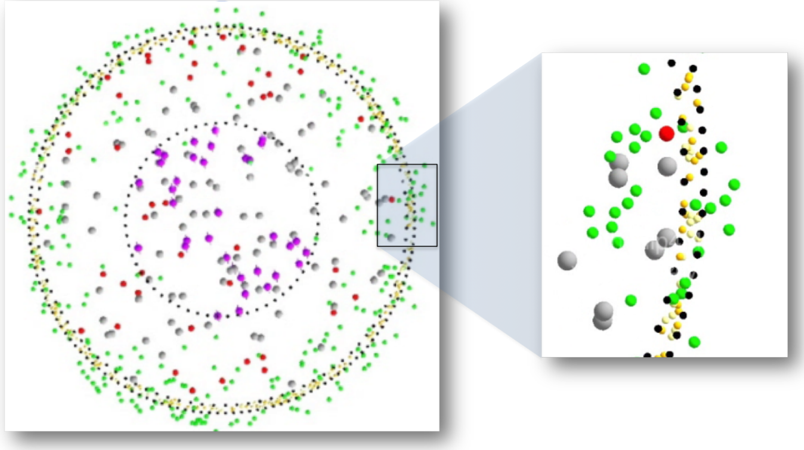

Fig. 1.1 Visual simulation of *E. coli* in FLAME: the cell (left side) and a zoomed-in snapshot of its membrane (right side)

Since we are interested only in some of the results of the FLAME simulations, in this section we consider a simplified version of *E. coli* cell, with only eight molecules types: dimers, regenerated dimers, monomers, oxygen, dimers bound to binding sites, *Cyo* proteins, *Cyd* proteins, binding sites, but without the *ArcBA* molecules. Note also that FLAME provides visualisation and animation capabilities, as seen in Figure 1.1, but we do not insist here on these aspects.

Oxygen molecules enter the cell through the membrane where a large amount of them are consumed by the oxidases (*Cyo* or *Cyd* proteins). Those which luckily get into the cell (about 1% of them) react with the *Fnr* dimers and generate *Fnr* monomers. Two *Fnr* monomers getting close enough could react and regenerate a *Fnr* dimer. The *Fnr* dimer which was already bound to binding sites can also be deactivated by oxygen molecules and leave the binding site. Also, *Fnr* dimers can bind to available binding sites when they get close enough.

Initially, the cell is considered to be in an anaerobic respiration state (no oxygen in the cell), and the values of the molecules are: 150 dimers, 1 monomer, 35 dimers bound to binding sites, 200 *Cyo* proteins, 200 *Cyd* proteins, 35 binding sites. In these conditions, FLAME can be used to simulate the reaction in the cell at different levels of oxygen, e.g., 100, 200, or 300 molecules. Figure 1.2 provides the simulation results for 100 molecules of oxygen over 10,000 iterations.

Besides the relation between the numbers of oxygen, *Fnr* dimer and *Fnr* monomer molecules, more interesting results are obtained for the model that also contains the *ArcBA* system. These are related to:

- the number of *ArcA* octamer, *ArcA* tetramer, *ArcA* dimer and *ArcB* (phosphorylated and non-phosphorylated);

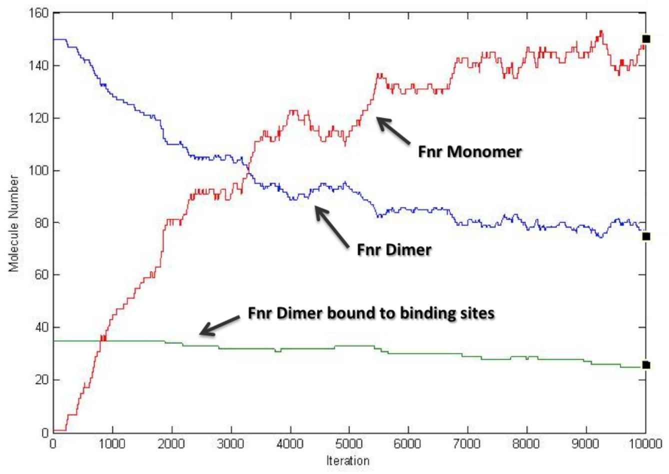

Fig. 1.2 Trend of molecule numbers with 100 oxygen molecules over 10,000 iterations

- the variation of the numbers of *Fnr* dimer and *ArcA* octamer which are bound to the binding sites, as these would determine the gene regulation and protein production.

So, as this model is currently used for simulation of biological processes, the number of the molecules (in biology, the concentration of certain gene regulator) is what the experiments are focusing on. More details on using FLAME for biological models, including the *E. coli* respiratory chain, can be found in [23].

1.5 A kernel P system corresponding to *E. coli*

In this section, we outline the way in which the previous X-machine based model is transformed into a kernel P system of degree 2 ($k\Pi$). The latter model is then used in the next section to formally verify some properties and simulate its behavior using the model checkers ProB and Spin. Due to the limitations of the corresponding modelling languages (e.g., the well-known state explosion problem) we considered a simplified model described below.

First of all, we split the *E. coli* cell into sectors (of circles) and we associate to each molecule a sector, depending on its coordinates. Thus, the objects of our P systems are pairs (l,s), where l is the label of the object and s is the corresponding sector. The idea of using topological spaces as control mechanisms for rule applications, in addition to the membranes themselves (thus offering a higher level of granularity), was introduced in [14]. As a consequence, the alphabet of $k\Pi$ is: $V = \{(Oxy_i, s_i) \mid i = 1..noOxy\} \cup \{(Cyd_i, s_i) \mid i = 1..noCyd\} \cup \{(Cyo_i, s_i) \mid i = 1..noCyo\} \cup \{(Dim_i, s_i) \mid i = 1..noDim\} \cup \{(BDim_i, s_i) \mid i = 1..noBDim\} \cup \{(BSite_i, s_i) \mid$

$i = 1..noBSites\} \cup \{(Mon_i, s_i) \mid i = 1..noMon\}$, where *Oxy* corresponds to the Oxygen molecules, *Dim* corresponds to the Dimers, *BDim* corresponds to the dimers bound to binding sites, *BSite* corresponds to the binding sites, *Mon* corresponds to the monomers and *noX* is the number of molecules of type X.

We assume that two molecules react only if they are in the same sector.

Then, the rules associated with the first compartment are:

- rewriting and communication rules corresponding to oxidations and oxygen transfer respectively:
  $(Oxy_i, s_i)(Cyo_j, s_j) \xrightarrow{p=99\%} \lambda \; \{s_i = s_j\}$
  $(Oxy_i, s_i)(Cyo_j, s_j) \xrightarrow{p=1\%} [(Oxy_i, s_i)]_2(Cyo_j, s_j) \; \{s_i = s_j\}$
  $(Oxy_i, s_i)(Cyd_j, s_j) \xrightarrow{p=99\%} \lambda \; \{s_i = s_j\}$
  $(Oxy_i, s_i)(Cyd_j, s_j) \xrightarrow{p=1\%} [(Oxy_i, s_i)]_2(Cyd_j, s_j) \; \{s_i = s_j\}$
- movement rules:
  $(Oxy_i, s_i) \rightarrow (Oxy_i, s_j) \; \{s_j = s_i - 1 \mid s_j = s_i \mid s_j = s_i + 1\}$
  $(Cyo_i, s_i) \rightarrow (Cyo_i, s_j) \; \{s_j = s_i - 1 \mid s_j = s_i \mid s_j = s_i + 1\}$
  $(Cyd_i, s_i) \rightarrow (Cyd_i, s_j) \; \{s_j = s_i - 1 \mid s_j = s_i \mid s_j = s_i + 1\}$

Probabilities associated with the above rules are required because, as we specified in the previous section, just about 1% of the oxygen molecules get into the cell. Thus, an oxygen molecule Oxy_i situated in a sector s_i in the first compartment is transferred in the second compartment with a probability of 1% or react with an oxidant situated in the same sector with a probability of 99%. These probabilities do not appear explicitly in the FLAME model because the oxidations occur more frequently due to the molecules location (the oxidases *Cyo* and *Cyd* act like a barrier for the oxygen).

The second compartment has the following rewriting rules:

- reactions between a dimer and an oxygen resulting two monomers
  $(Oxy_i, s_i)(Dim_j, s_j) \rightarrow (Mon_k, s_i)(Mon_l, s_i) \; \{s_i = s_j\}$
- reactions between a bounded dimer and an oxygen resulting a dimer and a binding site
  $(Oxy_i, s_i)(BDim_j, s_j) \rightarrow (Dim_k, s_i)(BSite_l, s_i) \; \{s_i = s_j\}$
- reactions between two monomers resulting a dimer
  $(Mon_i, s_i)(Mon_j, s_j) \rightarrow (Dim_k, s_i) \; \{s_i = s_j\}$
- reactions between a dimer and a binding site resulting a bounded dimer
  $(Dim_i, s_i)(BSite_j, s_j) \rightarrow (BDim_k, s_i) \; \{s_i = s_j\}$
- movement rules:
  $(Oxy_i, s_i) \rightarrow (Oxy_i, s_j) \; \{s_j = s_i - 1 \mid s_j = s_i \mid s_j = s_i + 1\}$
  $(Dim_i, s_i) \rightarrow (Dim_i, s_j) \; \{s_j = s_i - 1 \mid s_j = s_i \mid s_j = s_i + 1\}$
  $(Mon_i, s_i) \rightarrow (Mon_i, s_j) \; \{s_j = s_i - 1 \mid s_j = s_i \mid s_j = s_i + 1\}$
  $(BDim_i, s_i) \rightarrow (BDim_i, s_j) \; \{s_j = s_i - 1 \mid s_j = s_i \mid s_j = s_i + 1\}$
  $(BSite_i, s_i) \rightarrow (BSite_i, s_j) \; \{s_j = s_i - 1 \mid s_j = s_i \mid s_j = s_i + 1\}$

For all the above rules, guards are used to impose the constraints that two molecules react iff they are in the same sector ($s_i = s_j$) and, when moving, each molecule can remain in its sector ($s_j = s_i$) or can move into one of the neighboring sectors ($s_j = s_i - 1$ or $s_j = s_i + 1$).

Using these ideas and the particularities of each language, we constructed Event-B and a Promela models of $k\Pi$, and used them for formal verification. The obtained results are given in the next subsections.

1.6 Modelling, simulation and verification

1.6.1 Implementation in Event-B for ProB

Introduced by J.-R. Abrial [6], Event-B is a formal modelling language used for developing mathematical models of complex systems with a discrete behavior. It is supported by a platform called Rodin that integrates theorem-proving, model checking (ProB), and animation facilities.

The Event-B model of the kernel P system representation of *E. coli* described in the previous section is based on functions, operations with sets and non-deterministic assignments. Thus, for every set of molecules, we consider a partial function between a set of labels and the set of sectors. In the initialization event, every molecule initially in the *E. coli* cell is associated with its corresponding sector. The model contains an event for every reaction (rule), with the guards asking to the reactants to be in the same compartment, and the actions modifying the corresponding functions accordingly.

As an example, the event corresponding to the rule
$(Oxy_i, s_i)(Dim_j, s_j) \rightarrow (Mon_k, s_i)(Mon_l, s_i)\ \{s_i = s_j\}$ is:

```
Event DimerOxygen
any x, y, z1, z2
where
guard1: x ∈ dom(Oxy)
guard2: y ∈ dom(Dim)
guard3: Oxy(x) = Dim(y)
guard4: z1 ∈ ℕ\dom(Mon)
guard5: z2 ∈ ℕ\dom(Mon)
guard6: z1 ≠ z2
then
action1: Mon := Mon ∪ {z1 −> Oxy(x), z2 −> Oxy(x)}
action2: Oxy := Oxy\{x −> Oxy(x)}
action3: Dim := Dim\{y −> Dim(y)}
```

So, if the oxygen molecule x and the dimer y are in the same compartment (`guard3`), then two new monomers (z_1 and z_2) are produced and added to the monomers set (`action1`). Then, the reactants are consumed (`action2` and `action3`). The events for the other reactions are quite similar.

Probabilities associated with some rules are implemented using a random number generator, e.g., a random number positive integer N is generated and the rule with the probability of 1% is applied if N is divisible by 100 and the complementary rule, otherwise.

A different type of event is the one in which the molecules are moving. We remind that any molecule can non-deterministically remain in its sector or can move into one of the neighbouring sectors.

One of main problems in model checking is the state space explosion. In order to mitigate this, we considered in our model a variable called *state* with three possible values: *Reacting*, *Moving* and *Crash*. The system is usually in one of the states *Reacting* or *Moving*. The state *Crash* is considered in order to keep the number of configurations under control. When the system reaches a number of steps (*Max*), then its state becomes *Crash* and the verification stops. We considered in our experiments different values for *Max*, depending on the complexity of the property.

The results of verifying different properties based on this Event-B model are provided in Subsection 1.6.4.

A very important Event-B concept is refinement, which allows a model to be developed gradually [7]. A refined model contains more details about the system than the initial model and it is obtained by refining machines or extending contexts. Using this technique we can add to the previous Event-B model of *E. coli* the *ArcBA* system, i.e., variables corresponding to the *ArcBA* molecules and events corresponding to the reactions between them. We describe in the following how the refined model is obtained.

As in the initial model, we consider a partial function between a set of labels and the set of sectors for every set of *ArcBA* molecules. The initialization event has to be refined, associating to every new molecule the corresponding sector. The refined model contains seven new events, corresponding to the seven reactions that involve *ArcBA* molecules described in Section 1.3. Each of these is similar to those in the initial model. The event dedicated to molecules movement is also refined, the *ArcBA* molecules being also able to remain in the same sector or to move in a neighboring one. Even if the limitations of ProB, due again to state explosion, do not allow us to verify significant properties for the refined model, refinement proves to be an useful technique in building Event-B models of complex systems.

1.6.2 Implementation in Promela for Spin

This subsection describes the Promela implementation of the kP system model for *E. coli*. A mature model checking tool, Spin has been also successfully used in the context of membrane computing for verifying various types of P systems [27], including kernel P systems [20].

Although computationally equivalent with the investigated kP system model, the Promela implementation takes advantage of the power and flexibility of its specific modelling language constructs. For instance, a molecule is defined by using the type definition where the specific features are the *sector* where the molecule finds itself and a flag denoting whether it is still *active* or *consumed*.

```
typedef Molecule {
  int sector;
  bool isConsumed = true; }
```

Using this construct, each molecule type is assigned a global array representing the set of molecules of the same type.

The reactions between specific molecules are modelled using Promela macro-definitions. For instance, the kP system rule

$$(Mon_i, s_i)(Mon_j, s_j) \to (Dim_k, s_i)\{s_i = s_j\}$$

is computationally equivalent with the following Promela code:

```
inline MonomerMonomer() {
  skip;
  d_step {
    int x, y, foundX, foundY;
    bool found;
    found = false;

    for(x: 0..MaxMonCount-1) {
      for(y: 0..MaxMonCount-1) {
          if
          :: !found && Mon[x].isConsumed == false && Mon[y].isConsumed ==
               false && Mon[x].sector == Mon[y].sector && x != y ->
                  found = true; foundX = x; foundY = y;
          :: else -> skip;
          fi; } }

    if
    :: found ->
        createNewDim(Mon[foundX].sector);
        Mon[foundX].isConsumed = true;
        Mon[foundY].isConsumed = true;
    :: else -> skip;
    fi;
    skip; } }
```

Another well-suited Promela feature for *E. coli* modelling is the nondeterministic control statement. Throughout the model, nondeterministic guarded commands are used for simulating random molecule assignment to regions, molecule movement and reaction triggering. For instance, the change in position of *Fnr* dimer molecules are modelled using the following macro definition:

```
inline moveDim() {
  int index;
  for(index: 0..MaxDimCount-1) {
    if
    :: Dim[index].isConsumed == false ->
      if
      :: true && Dim[index].sector > 0 -> Dim[index].sector--;
      :: true -> skip;
      :: true && Dim[index].sector < MaxSectorCount -> Dim[index].sector++;
      fi;
    :: else -> skip;
    fi; }
  index = 0; }
```

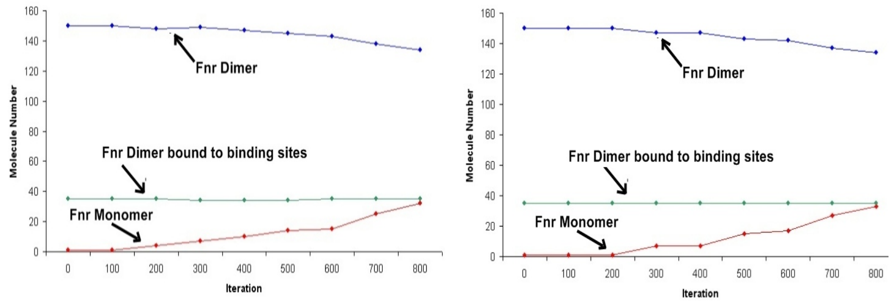

Fig. 1.3 Trend of molecule numbers with 100 oxygen molecules over 800 iterations: in Event-B (left) and in FLAME (right)

All the above macros are wrapped up in a scheduler process which is responsible for evolving the system from an initial configuration to a final one, with respect to a predefined number of steps. At each step, random molecule movement and reaction triggering is issued in order to simulate the nondeterministic behavior of *E. coli* components.

1.6.3 Simulation results

In this subsection we provide some simulation results obtained for the Event-B and Promela models of *E. coli*, by comparison with the FLAME simulation results.

The number of molecules of type X is denoted as noX. As for the FLAME model, the initial values of the molecules are: $noDim = 150$, $noMon = 1$, $noBDim = noBSite = 35$, $noCyd = noCyo = 200$. In these conditions, ProB can be used to simulate the events corresponding to the reactions in the cell at different levels of oxygen. Figure 1.3 provides an average of simulation results for 100 molecules of oxygen over 800 iterations. Although the current limitations of ProB do not allow us to simulate more steps in the evolution of the Event-B model, Figure 1.3 shows a similar variation of the molecule numbers for both models. The small differences between the two graphs are due to the approximations made in the kernel P system model.

Besides the Event-B simulations, a series of simulations have been conducted using the Promela model and Spin model checking tool, using the same initial configuration of molecule numbers and different levels of oxygen. The trend of molecule numbers was similar as for the simulations conducted using the Event-B and FLAME models. However, compared to ProB, Spin was able to run more iterations and performed better in terms of execution time and memory requirements.

1.6.4 Verification results

We present now the different properties that we verified, mainly checking how the number of different molecules evolves during the cell reactions.

We start again with the Event-B model and ProB model checker. In the Event-B model, we introduced variables counting the number of molecules of each type, denoting again with *noX* the number of molecules of type X. Initially, we considered $noOxy = noCyd = noCyo = 100$, $noMon = 1$, $noDim = 75$, $noBdim = noBSites = 18$.

Properties verified with ProB can be formulated using the LTL (linear temporal logic) formalism. We give some of these properties and the result given by the model checker in the following:

- $G\{noMon < 3$ or $state = Crash\}$; the model checker returns a counterexample so, in some situations, a dimer is divided in two monomers before reaching the state Crash.
- $G\{noMon < 7$ or $state = Crash\}$; no counterexample found, so the insertion rate of oxygen in the second compartment is very low.
- $G\{noBdim > 17\}$; the model checker returns a counterexample so the bounded dimers are sometimes involved in reactions.

Unfortunately, the current limitations of ProB do not allow us to verify more complicated properties or to increase the number of molecules of each type. Our future work will concentrate on improving these aspects. All the above properties were also verified with Spin, obtaining the same results. More than that, as we see below, the results obtained with the Spin model checker compensate for some cases not tackled by ProB.

In Spin, we have also used LTL to specify the investigated properties. In order to conduct the verification, we must take into account that the base model is a kP system and the properties must be verified in the context of a P system state, after the maximally parallel application of the rules in each compartment, and not in every state of the Promela model. In order to accomplish this, a special boolean variable, called *isPSystemStep*, is used for identifying P system states. This variable is included in the LTL formulae in order to instruct Spin to consider only states where the configuration reaches a relevant point in the system execution.

The following investigations have been conducted using an initial configuration of $noCyo = 200$, $noCyd = 200$, $noDim = 150$, $noMon = 1$, $noBdim = 35$, $noBSites = 35$, and the number of oxygen molecules varying between 100, 200 and 300, aiming to verify the relation between the number of different molecules and the evolution of the system after some key points in the reaction process:

- F ($isPSystemStep$ and $nrOxy$ = 0) $-$ using an initial configuration of $noOxy$ = 100, after 500 steps, $noOxy$ will eventually decrease to 0.
- G (($isPSystemStep$ and $noMon \leq 2 * noDim$) or $!isPSystemStep$) $-$ with initial $noOxy$ = 300, after 500 steps, the number of *Fnr* monomers will be at most half the number of *Fnr* dimers.

- G $(!(noOxy = 0$ and $prevNoOxy = 0)$ or $noMon \leq prevNoMon$ or $!isPSystemStep)$ $-$ after the point when $noOxy = 0$, no *Fnr* monomers will be produced, with an initial value of $nrOxy = 100$ and 200 execution steps.
- G $(!(noOxy = 0$ and $prevNoOxy = 0)$ or $noBSites \leq prevNoBSites$ or $!isPSystemStep)$ $-$ with an initial configuration of $nrOxy = 100$ and 200 execution steps, no more binding sites will become available after the point when $noOxy = 0$.
- G $(!isPSystemStep$ or $!(noOxy = 0$ and $prevNoOxy = 0$ and $noBSites = 0$ and $prevNoBSites = 0)$ or $noBDim = prevNoBDim)$ $-$ after reaching the state when $noOxy = 0$ and $noBSites = 0$, the number of bounded dimer molecules will remain unchanged, for an initial $noOxy = 200$, and $1,000$ execution steps.
- G $(!isPSystemStep$ or $!(noOxy = 0$ and $prevNoOxy = 0)$ or $noBDim \geq prevNoBDim)$ $-$ using initial 100 oxygen molecules, after 200 steps, the number of bounded dimer molecules will remain the same or at most increase, after reaching the state when $noOxy = 0$.
- G $(!isPSystemStep$ or $(noOxy = 0 \rightarrow$ F $(noBDim = 70$ and $isPSystemStep)))$ $-$ for the given configuration, all the existent binding sites will eventually become occupied, after the point when all the oxygen will be consumed, for initial $noOxy$ $= 100$ and $1,000$ execution steps.

For the set of properties for which a counterexample was issued, the verification time was up to 3 minutes. On the other hand, for the remaining properties, the time varied between 30 and 40 minutes, depending on the number of iterations and the complexity of the property.

The previous results were obtained by running the models on an Intel Xeon CPU with a speed of 2.4 GHz and 8 GB of RAM.

1.6.5 Event-B vs. Promela

As detailed in the previous subsections, the conducted simulations and verifications were supported by modelling the *E. coli* processes into Event-B and Promela formal specification languages and taking advantage of their mature tool support, Rodin and Spin, respectively. Despite both providing powerful modelling capabilities, the two languages are basically very different in their modelling approaches. Event-B models are abstract state machines in which transitions between states are implemented as *events*. An event is a state transition which is specified in terms of *guards* and *actions*. Guards are necessary conditions for an event to be enabled. Actions describe how the occurrence of an event modify some of the variables of the model. On the other hand, Promela provides a powerful set of instructions for describing *concurrent processes* and *inter-process communications*.

Each model takes advantage of their corresponding language constructs for implementing the necessary functionality. Having a formalism based on the set theory, the Event-B model uses functions, sets and set operators as building blocks for specifying the molecule evolution rules. The Promela model implements the corresponding functionality as a scheduling process for synchronizing and running a maximum number iterations for evolving the *E. coli* molecules starting from a given initial configuration. The non-deterministic conditional and cycling instructions available in Promela recommends it as a suitable specification language for modelling the non-deterministic behavior of the different molecules.

Despite the fact that both languages proved suitable modelling capabilities for in-silico *E. coli* simulations, the verification and simulation tasks performed better in case of Spin in terms of complexity of the properties being verified, memory and time, recommending it once again and increasing the confidence for being a leading model checking tool in its class.

1.7 Conclusions

A constant concern of biologists, the bacteria *E. coli* has an interesting behavior in relation to the level of oxygen in the environment. In this chapter, we built a (simplified) kernel P system model based on the FLAME model of *E. coli* and we complemented simulations with formal verifications that can check various patterns of behaviour on the model. For instance, such verification can provide ”sanity checks” on the model, which should increase the confidence of modellers and biologists that the models behaves as expected. Then, the kernel P system was implemented in two modelling languages, Event-B and Promela, and simulation results show that the kP system is consistent with the original FLAME model. Using the two implementations and the associated model checkers, several properties, formulated using the LTL formalism, were verified. Event-B proved to be more convenient for modelling, while Spin was more efficient for simulation and verification. The models and results used in this chapter are uploaded on the web page of the MuVeT project [2].

In order to complement our approach, we are currently investigating invariant property generation (using a tool called Daikon), which should help biologists to build a good list of properties to be verified on the model, thus increasing the quality of the used model. Our future work will also concentrate on finding strategies to verify more complicated properties using the model checkers, developing the model by adding the *ArcBA* molecules, and applying similar methodology to other biological entities.

Acknowledgements This work was partially supported by the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, via MuVeT project [2], code PN-II-ID-PCE-2011-3-0688.

References

1. FLAME web site. `http://flame.ac.uk`.
2. MuVeT web site. `http://muvet.ifsoft.ro/e-coli.html`.
3. ProB web site. `http://www.stups.uni-duesseldorf.de/ProB`.

4. Spin web site. `http://spinroot.com`.
5. SUMO project. `http://sysmo-sumo.mpi-magdeburg.mpg.de/trac/wiki/public`.
6. J.-R. Abrial. *Modeling in Event-B. System and software engineering*. Cambridge University Press, New York, USA (2010).
7. J.-R. Abrial, S. Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to Event-B. *Fundamenta Informaticae*, **77**, (2007), 1–28.
8. O. Andrei, M. Calder. Trend-based analysis of a population model of the AKAP scaffold protein. *Transactions on Computational Systems Biology*, **7625**, (2012), 1–25.
9. O. Andrei, G. Ciobanu, D. Lucanu. A rewriting logic framework for operational semantics of membrane systems. *Theoretical Computer Science*, **373**, 3, (2007), 163–181.
10. D. Benque, S. Bourton, C. Cockerton, B. Cook, J. Fisher, S. Ishtiaq, N. Piterman, A. Taylor, M. Vardi. BMA: Visual tool for modeling and analyzing biological networks. In *Proc. CAV'12*, volume 7358 of *LNCS*. Springer (2012), pages 686–692.
11. F. Bernardini, M. Gheorghe, F. J. Romero-Campero, N. Walkinshaw. A hybrid approach to modeling biological systems. In *WMC 2007*, volume 4860 of *LNCS*. Springer (2007), pages 138–159.
12. J. Blakes, J. Twycross, F. J. Romero-Campero, N. Krasnogor. The Infobiotics workbench: an integrated *in silico* modelling platform for systems and synthetic biology. *Bioinformatics*, **27**, 23, (2011), 3323–3324.
13. G. Ciobanu, M. J. Pérez-Jiménez, G. Păun, editors. *Applications of Membrane Computing*. Natural Computing Series. Springer (2006).
14. E. Csuhaj-Varjú, M. Gheorghe, M. Stannett. P systems controlled by general topologies. In *Proc. UCNC'12*, volume 7445 of *LNCS*. Springer (2012), pages 70–81.
15. D. Díaz-Pernil, C. Graciani, M. Gutierrez-Naranjo, I. Pérez-Hurtado, M. Pérez-Jiménez. Software for P systems. In Gh. Păun, G. Rozenberg, A. Salomaa, editors, *The Oxford Handbook of Membrane Computing*. Oxford University Press (2010), pages 118–143.
16. D. Díaz-Pernil, I. Pérez-Hurtado, M. Pérez-Jiménez, A. Riscos-Núñez. A P-Lingua programming environment for membrane computing. In *Proc. WMC'08*, volume 5391 of *LNCS*. Springer (2008), pages 187–203.
17. S. Eilenberg. *Automata, languages and machines*. Academic Press, New York (1994).
18. M. Gheorghe, F. Ipate, C. Dragomir. A kernel P system. In *Proc. BWMC10*. Fénix Editora (2012), pages 153–170.
19. M. Gheorghe, F. Ipate, R. Lefticaru, C. Dragomir. An integrated approach to P systems formal verification. In *Proc. CMC'10*. ProBusiness Verlag (2010), pages 225–238.
20. M. Gheorghe, F. Ipate, R. Lefticaru, M. J. Pérez-Jiménez, A. Turcanu, L. Mierla, L. Valencia Cabrera, F. M. Garcia-Quismondo. 3-Col problem modelling using simple kernel P systems. *International Journal of Computer Mathematics*, **90**, 4, (2013), 816–830.
21. A. Hinton, M. Z. Kwiatkowska, G. Norman, D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. TACAS'06*, volume 3920 of *LNCS*. Springer (2006), pages 441–444.
22. M. Holcombe. X-machines as a basis for dynamic system specification. *Software Engineering Journal*, **3**, 2, (1988), 69–76.
23. M. Holcombe, et al. Modelling complex biological systems using an agent-based approach. *Integr. Biol.*, **4**, (2012), 53–64.
24. G. Holzmann. The Model Checker SPIN. *IEEE Trans. Softw. Eng.*, **5**, 23, (1997), 279–295.
25. F. Ipate, T. Balanescu, P. Kefalas, M. Holcombe, G. Eleftherakis. A new model of communicating stream X-machine systems. *Romanian Journal of Information Science and Technology*, **6**, 1-2, (2003), 165–184.
26. F. Ipate, R. Lefticaru, L. Mierla, L. Valencia Cabrera, H. Han, G. Zhang, C. Dragomir, M. J. Pérez-Jiménez, M. Gheorghe. Kernel P systems: Applications and implementations. In *Proc. BIC-TA'13*, volume 202 of *Advances in Intelligent Systems and Computing*. Springer (2013), pages 1081–1089.
27. F. Ipate, R. Lefticaru, C. Tudose. Formal verification of P systems using SPIN. *Int. Journal Found. Computer Science*, **22**, 1, (2011), 133–142.

28. F. Ipate, A. Turcanu. Modelling, verification and testing of P systems using Rodin and ProB. In *Proc. BWMC9*. Fénix Editora (2011), pages 209–220.
29. M. Kwiatkowska, G. Norman, D. Parker. *Symbolic Systems Biology*, chapter Probabilistic Model Checking for Systems Biology. Jones and Bartlett (2010), pages 31–59.
30. M. Leuschel, M. Butler. ProB: An automated analysis toolset for the B method. *International Journal on Software Tools for Technology Transfer*, **10**, 2, (2008), 185–203.
31. A. Obtulowicz, G. Paun. (In search of) Probabilistic P systems. *Biosystems*, **70**, 2, (2003), 107–121.
32. I. Pérez-Hurtado, L. V. Cabrera, M. J. Pérez-Jiménez, M. A. Colomer. MeCoSim: A general purpose software tool for simulating biological phenomena by means of P systems. In *Proc. BIC-TA'10*. IEEE Xplore (2010), pages 637–643.
33. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, **61**, 1, (2000), 108–143.
34. G. Păun, G. Rozenberg, A. Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press (2010).
35. P. Richmond, D. Walker, S. Coakley, D. Romano. High performance cellular level agent-based simulation with FLAME for the GPU. *Briefings in Bioinformatics*, **11**, 3, (2010), 334–347.
36. A. Turcanu, F. Ipate. Modelling, testing and verification of P systems with active membranes using Rodin and ProB. In *Proc. CMC'11*. Paris-Est University Press (2011), pages 459–468.